Cromemco™
CROMIX*

Multi-user Multi-tasking

DISK OPERATING SYSTEM

INSTRUCTION MANUAL

CROMEMCO, Inc.
280 Bernardo Avenue
Mountain View, CA 94043

Part No. 023-4022

August 1980

*trademark of Cromemco, Inc.

This manual was produced on a Cromemco System Three computer utilizing a Cromemco HDD-22 Hard Disk Storage System running under the Cromemco Cromix Operating System. The text was edited with the Cromemco Cromix Screen Editor. The edited text was formatted using the Cromemco Word Processing System Formatter. Final camera-ready copy was printed on a Cromemco 3355A printer.

Table of Contents

Introduction to the Cromemco Cromix Operating System

When microcomputers were first introduced, the most common memory modules contained four thousand bytes of storage. Today sixteen times as much memory is available in a module. Today's microcomputers also utilize the new technology incorporated in fast hard disk mass storage devices.

The Cromix Operating System was developed by Cromemco to fully take advantage of the large amount of random access memory (RAM) and fast hard disk storage available on today's and tomorrow's microcomputers. The Cromix Operating System has many capabilities only found in large mainframe operating systems, and then some - Capabilities such as the following:

1. support of multiple tasks and multiple users on hard disk and floppy disk file storage systems,

2. multiple hierarchical directories and sub-directories,

3. compatible I/O which supports user redirection of input and output,

4. versatile Shell program for flexible and reconfigurable user interface,

5. password security system limiting system and file access as well as protecting files with read, write, append, and execute attributes,

6. date and time support,

7. numerous file buffers for high speed execution, and

8. resident, swapping-free execution of tasks and servicing of users through bank selection for rapid context switching.

A Cromemco customer has a choice of using either the Cromix Operating System or CDOS on the Cromemco microcomputers. CDOS has the advantage of years of testing by thousands of users. It is a time proven system. In addition CDOS has the advantage of being compact in memory utilization. It can reside in the same sixty four thousand byte memory board as the user. Only sixty four thousand bytes of RAM memory is required for CDOS, and CDOS uses only about one fourth of that memory with the rest available for programming languages and user programs.

The Cromix Operating System requires sixty four thousand bytes of RAM for the Operating System. Each concurrently executing program requires an additional 64K bytes, of which only 1K is used by the Operating System. However, CDOS supports only one directory, only one user and task, does not support date and time or offer password security and is not reconfigurable in I/O or user interface. The Cromix Operating System offers all of these features. Additionally, CDOS offers limited buffering because of its small memory size. The extensive buffering of the Cromix Operating System makes disk-intensive execution more than twice as fast as CDOS.

Some of these features may not be familiar to many computer users. CDOS may have all of the features that many users expected before the advent of the Cromix Operating System. It may be difficult to imagine the ability of the Cromix Operating System to print a file at the same time as the user is editing another file unless you have used a computer that offered that capability. The Cromix Operating System allows you not only to print but to execute multiple jobs from one or several terminals at the same time. This multi-processing is commonplace on large mainframe computers. So are the time and security features of the Cromix Operating System.

Not common on large mainframes is the ability to allow the user to reconfigure the I/O and user interface. Disk files may be used in the place of a keyboard for pre-programmed responses to standard programs. Disk files may also be used to store program output sent to the user terminal screen. If a user does not expect to be present at the terminal during execution of a program, the output may be redirected to a disk file for later viewing. The user interface may be radically changed when using the Cromix Operating System. Usually an operating system does not allow the user to change commands but the Cromix Operating System has a programmable Shell which facilitates user interface customization. Thus a user should expect to use the Cromix Operating System wherever productivity can be increased by utilizing the ability of the computer to perform multiple tasks at the same time. Some users will find the greater disk throughput of the Cromix Operating System or the support of multiple directories and sub-directories alone justifies its use. For whatever reason the Cromix Operating System is chosen, the user will have access to features that are truly at the state of the art of operating systems and yet are easy to learn and use.

Getting Started

This chapter is intended to be an introduction to the Cromemco Cromix Operating System for the first-time user. By leading the reader through an exemplary session, many of the important features of the Cromix Operating System will be highlighted. The reader is encouraged to go through this chapter while sitting in front of a terminal and to mimic and expand on the examples given herein. By doing this several times the novice user should be brought to a level of competence which will allow a fuller understanding of the balance of the manual.

Initial hardware and software setup are covered in other sections. It is assumed here that the hardware is set up and functioning properly and that the user has been assigned a user name.

Login

Because the Cromix Operating System can serve many different users, and because each user may have access to a unique set of files, a valid user name must be presented to the System before the user can be logged in. Please refer to the section of this manual entitled Setting Up - Software if it is necessary to establish a new user name. For this example it is assumed that the user name **fred**, with the secret password **mountain** has been previously established.

In response to the Cromix System prompt Login:, the user must respond with a valid user name and password:

```
     Login: fred<CR>
     Password:
```

Notice that when the user types the secret password it is not displayed on the terminal. After the password and the following **RETURN** have been entered, the Cromix Operating System will respond:

```
     Logged in fred Jun-24-1980 17:12:15 on console
     %
```

Throughout this manual messages and prompts displayed by the Cromix Operating System will be typed normally (and sometimes underlined for clarity) while user-supplied responses will be typed in **bold face** characters. Depressing the

RETURN key on the terminal will be represented by **<CR>**. Thus, in the above example the Operating System has displayed the prompt login: and the user has supplied the response **fred** followed by a carriage return.

If a valid user name is provided, the Cromix Operating System will respond by displaying the message of the day (motd) and a prompt. The prompt is either a percent sign (%) or a pound sign (#). When the prompt is displayed, it indicates that the Operating System is waiting for further instructions.

Important Note

The Cromix Operating System is configured so that information will not scroll off of the terminal screen before the user has had a chance to review it. When the screen is full the Operating System will cause the terminal to emit a **beep.** The user should enter a **CNTRL-Q** to indicate to the Operating System that the information on the screen is no longer needed. A **CNTRL-Q** is entered by holding down the **CNTRL** (on some terminals its **CTRL**) key and simultaneously typing **Q.**

This feature can be disabled by running the **mode** utility as follows:

```
    % mode -pa<CR>
```

The important thing to remember is, if the terminal seems to have locked up, type **CNTRL-Q.**

Logging Off

A user may log off of the system by entering **ex** or **exit** in response to the Cromix Operating System prompt.

Editing Files

As a first exercise we will create a file containing a list of names. This involves using the Screen Editor which will be covered very briefly here. For further information the reader is referred to the Cromemco Screen Editor Manual (part number 023-0081) and to the Screen Editor Utility which is covered in the Utilities section of this manual.

The following command will cause the Operating System to load the Screen Editor and create the desired file:

```
% screen friends<CR>
```

If everything is working properly, the banner for the Screen Editor will be displayed momentarily and then the console will be cleared and the Screen Editor prompt will be displayed across the top of the screen.

For this example we will write a list of names to the **friends** file by using the Screen Editor. This is done (once the Screen Editor has been called) by typing **i** (for insert) followed by the desired list of names, each terminated by a **<CR>**. The **ESC**ape key must then be depressed to indicate to the Screen Editor that we are no longer inserting text. Finally, the commands are given to exit from the Screen Editor and write (update) the friends file by typing the characters **e** (for exit) and **u** (for update). The number of characters written to the **friends** file will be displayed followed by the Cromix Operating System prompt.

The type command may be used to display the file which was just created:

```
% type friends<CR>
```

File System Structure

The Cromix File System may be thought of as an upside-down tree. At the top of the tree we have the root and coming down from the root are the branches. Some of the branches have additional branches as offshoots. Some do not. Note that our tree has no trunk, the branches grow directly out of the root.

Node is the term which is used to refer to those places on the tree where a branch separates into one or more additional branches. Node is also used to refer to the tips of the branches. In the Cromix File System every node has a name.

Having established a tree, and having named each of the nodes of the tree, let us suppose that it becomes necessary to give someone directions to climb out to a specific branch of the tree. The directions will instruct the climber to start at the location where two or more branches separate off from the root. This location is still called the root. From this location our climber is directed to a node. From this node the climber may be directed to an adjacent node. The climber can only climb between nodes which are connected by branches. This process continues until the climber has reached the desired node. By using this method we can instruct the climber to climb to any specific tip of a branch or intersection where one or more branches are joined.

The instructions can be simplified into a list of nodes given in the order which the climber will reach them. The term **path name** is used to refer to this list of nodes.

There are two additional things which must be established to make our analogy complete. The first is that the nodes which have additional branches coming off them are called **directory** nodes. A directory node has its own name, as do all nodes. In addition, a directory node contains a list of all of the names of the nodes which will be found at the end of its branches, thus the term directory.

The second is that the nodes at the tips of each of the branches are called **ordinary** nodes or ordinary files. Any type of information may be stored at an ordinary node.

And so the analogy ends. The tree is the file structure which the Cromix Operating System uses to store its files. The root is the root directory which is always present. Under the Cromix Operating System the root directory is always named /. The directory nodes contain pointers to other directories and ordinary files. The user stores information in these ordinary files. The ordinary files may contain programs, text, or data.

The Cromix Operating System locates a given directory or ordinary file by use of a path name. A path name which is used to locate a directory is called a **directory path name.** A path name which is used to locate an ordinary file is called a **file path name.**

Path Names

Although path names do not need to start with the root directory we will confine ourselves to this type of path name for the current discussion. A path name traces a path from the root directory, through any intermediate directories, to the desired directory or file. For example, the file path name for the file **motd** is:

```
/etc/motd
```

The initial / in the file path name specifies the root directory. Each subsequent / in a path name separates entries of the path name. The next entry in the above path name is **etc** which is another directory. Another / separates this directory from an entry in the directory. This entry is **motd** which is an ordinary file. (Refer to Figure 1.)

If a file name is included in a path name, it must be the last entry in the path name. This is called a file path name. Refer to Figure 2 and trace the following file path name:

```
/letters/business/abcompany/thomas
```

Refer to Figure 3 and trace the following directory path name:

```
/letters/personal/rose/june18
```

A file path name may be used anywhere the Cromix Operating System expects a file name. Similarly, a directory path name may be used anywhere a

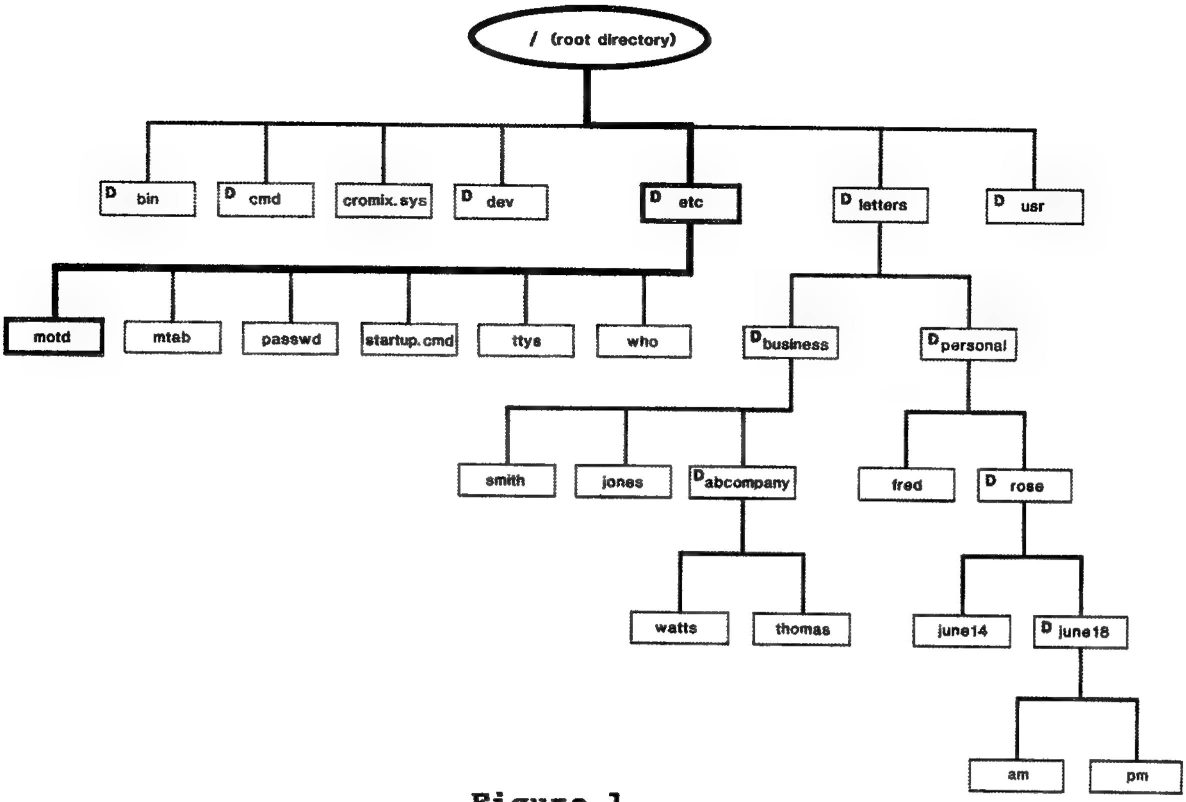


Figure 1

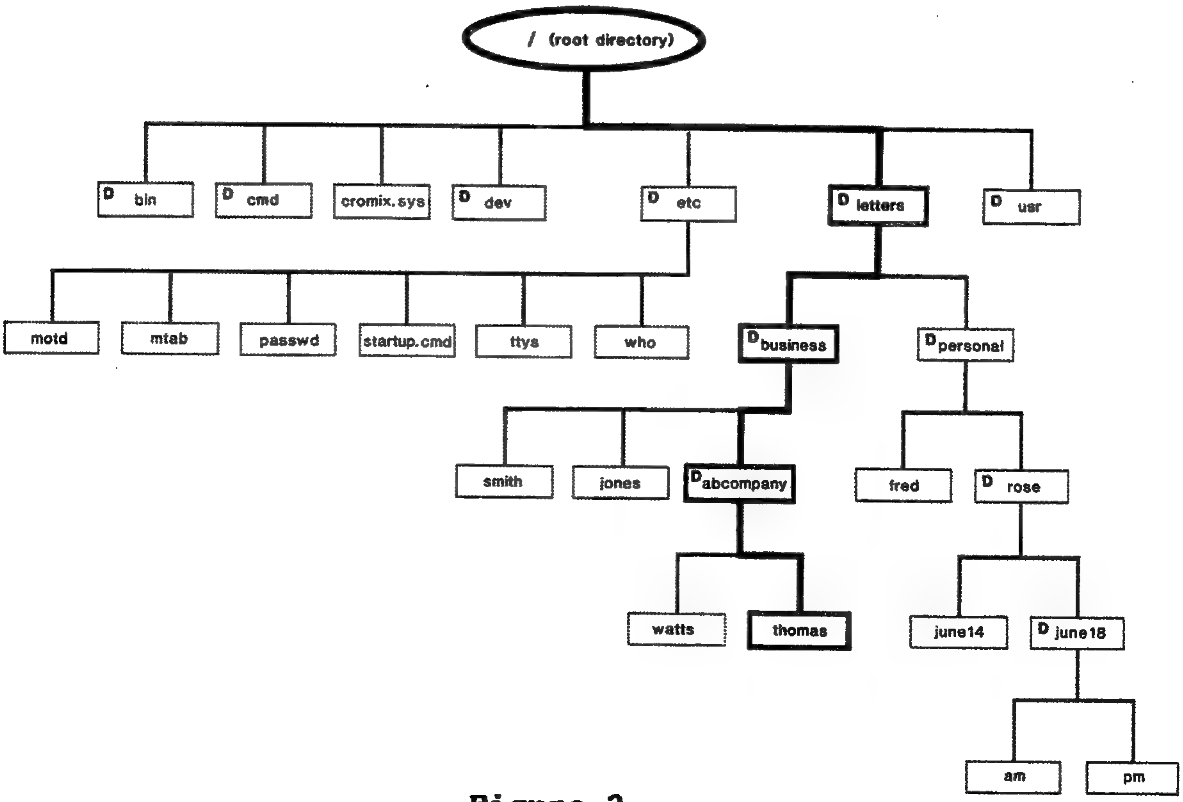


Figure 2

directory name is expected.

Current Directory

The current directory specifies those files and directories which can be accessed by only a file or directory name (i.e., no path name is needed). The user has immediate access to the current directory; any other directory must be explicitly specified on the command line.

The current directory can be thought of as just another directory to start a path name. The advanced user is referred to the discussion of relative path names in the Advanced Features section of this manual.

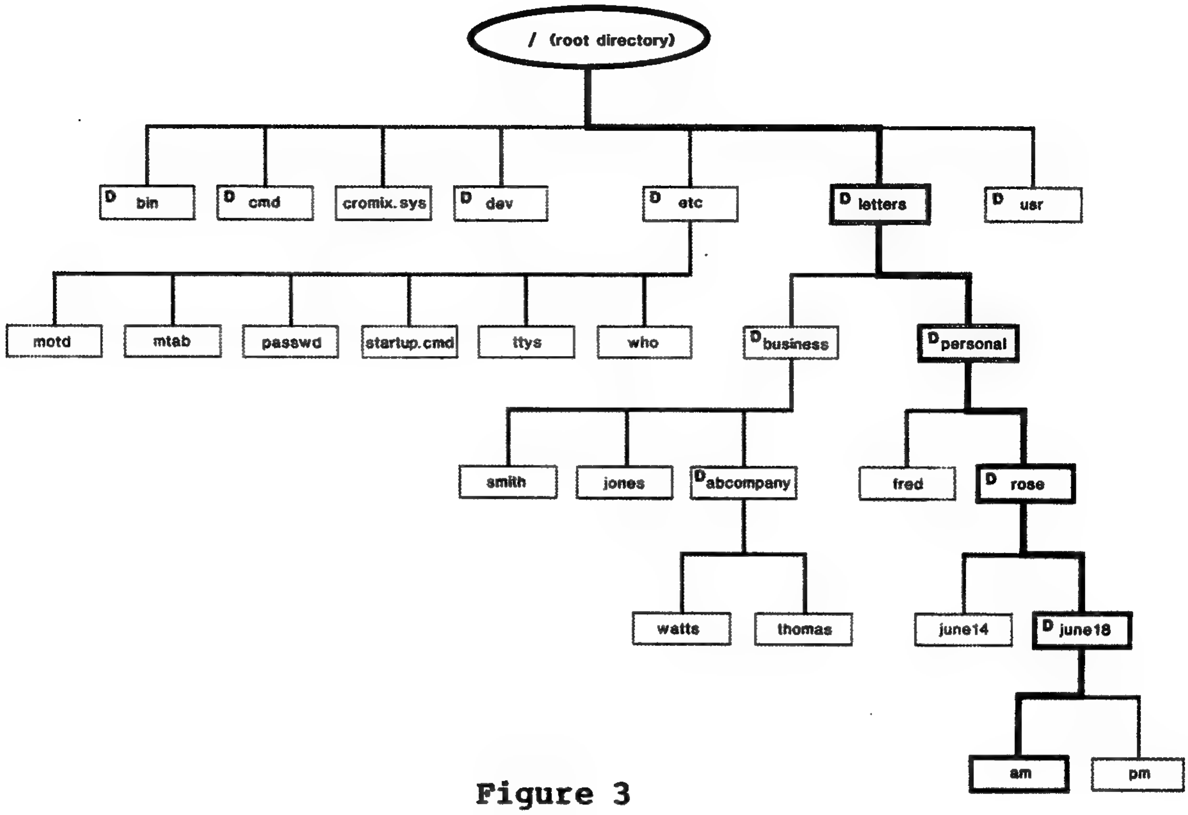

/ (root directory)
D bin
D cmd
cromix.sys
D dev
D etc
D letters
D usr
motd
mtab
passwd
startup.cmd
ttys
who
D business
D personal
smith
jones
D abcompany
fred
D rose
watts
thomas
june14
D june18
am
pm

Figure 3

The Basic Commands & Utilities

This section will cover the use of some of the commands and utilities which are required when using the Cromix Operating System. More complete descriptions of these and other commands and utilities are given later in the manual.

Directory Command

When the **d**(irectory) command is given, the Cromix Operating System will respond by displaying the name of the current directory:

```
% d<CR>
/
%
```

In the above example the Operating System responded to the command to display the name of the current directory by displaying / which is the name of the root directory. The d command can also be used to change the current directory:

```
% d /etc<CR>
% d<CR>
/etc
%
```

The Operating System does not acknowledge the successful completion of the command to change the current directory. In the above example, the user changed the current directory and then entered the **d** command to determine if the current directory had indeed been changed. The Cromix Operating System responded by displaying the name of the current directory, /etc.

List Utility

The utility which is used to display an alphabetical list of entries in a directory is **l**(ist):

```
% l
        1,216 D  1 bin
           64 D  1 cmd
       36,864    1 cromix.sys
          448 D  1 dev
          352 D  1 etc
           32 D  1 letters
          192 D  1 usr
```

In response to the **l** command above, the Operating System displayed a list of all the sub-directories and ordinary files contained in the current directory.

The **l** command displays four columns of information. The column on the left is the number of bytes occupied by the file or directory. The second column will be blank if the entry is an ordinary file and will contain a **D** if the entry is a directory. The third column indicates the number of links to the given directory or file. (Links will be covered in a subsequent section.) The column on the far right is the name of the entry which is either a directory or an ordinary file.

There are several ways to command the Cromix Operating System to list the entries within a given directory. First, using the **d** command, we can make the directory in question the current directory and then, using the **l** utility, list the contents of the current directory:

```
# d /etc
# l
    .
    .
    .
```

Another way to list the entries in a directory (which is not the current directory) is to give the **l** command together with the directory path name:

```
# l /etc
    .
    .
    .
```

Make Directory Command

Let us establish part of the file system shown in Figure 3. The first step in this process is to create the necessary directories. This is done by using the make directory command, **makdir**.

```
% makdir /letters
% makdir /letters/personal
% makdir /letters/personal/rose
% makdir /letters/personal/rose/june18
```

In the above example the user has created four new directories. Each of these directories is a sub directory of the previously created directory.

With careful planning this type of file structure allows the user to organize great numbers of files so that each is readily accessible.

We can now use the Screen Editor to create a file named **am** located in the directory named **June 18:**

```
% screen /letters/personal/rose/june18/am
```

If you are going to be doing quite a bit of work in a given directory, it is easier to change the current directory rather than specify a long path name every time a file is used. This can be done by giving the directory command with the desired directory path name:

```
% d /letters/personal/rose/june18
% d
/letters/personal/rose/june18
%
```

In the above example the user used the directory command first to change the current directory and then to display the path name of the current directory. Now it is a simpler matter to call the Screen Editor and request that the file **am** be created in the current directory:

```
% screen am
```

It is left to the reader to use the Screen Editor to create a file named **am** in the aforementioned directory. Users who are not familiar with the Screen Editor are referred to the Cromemco Screen Editor manual.

Note that all directories specified in a path name must have been previously created with the **makdir** command. The Cromix Operating System will not automatically create directories.

Type Command

Let us assume that the file **am** exists as specified in Figure 3. The **type** command may be used to display the contents of the file. Notice that the type command may be abbreviated **ty.**

```
% ty /letters/personal/rose/june18/am
```

If the full file path name (starting with /, the root directory) is used, we may review a file in another directory without changing the current directory. Similarly, we may edit a file in another directory without changing the current directory.

Rename Command

Rename, abbreviated **ren,** must be followed by the existing name (or path name), a space, and the new name (or path name). For example:

```
% ren fred joe
% ren /letters/business/jones /letters/business/william
```

In the example above, the file named **fred** was renamed **joe.** The file was in the current directory so that no path name was used. The second example involves a file which was not located in the current directory. The name of the file was changed from **jones** to **william.** Because the file was not in the current directory, the entire path name for the file was specified.

Delete Command

The delete command, abbreviated **del,** is followed by the name (or path name) of the file or directory which is to be deleted. In order to delete a directory, all of the files in the directory must have been previously deleted and it must not be the current directory. The contents of a file which has been deleted may not be recovered.

```
% del joe
% del /letters/business/william
```

In the above examples, the files which were renamed in the discussion of the **ren** command were deleted.

Advanced Features

Tree Data Structure

A tree is a data structure. A data structure is a method of storing data or information so that it may be easily accessed. As will be seen from Figure 4, the tree data structure is the inverse of a natural tree.

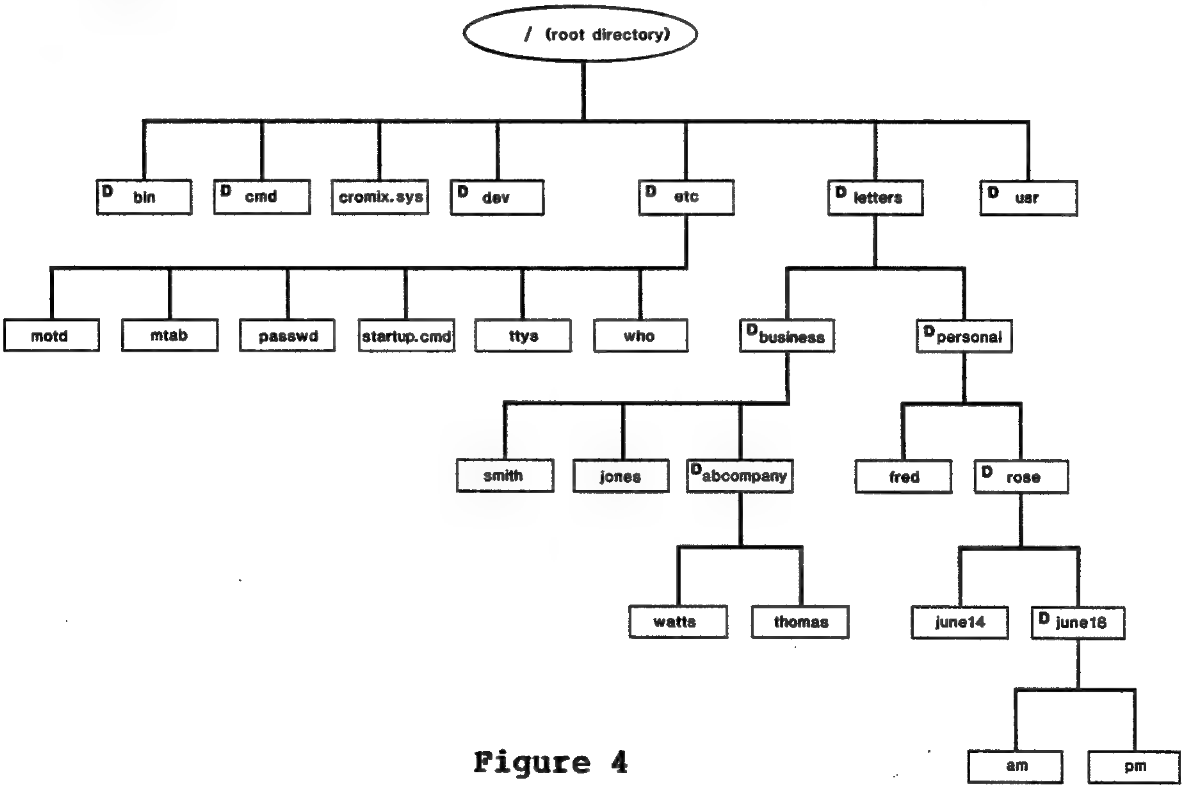


Figure 4

The root is at the top and the branches grow down from the root. In addition to branches and a root, the tree data structure has nodes. The term node is given to any one of the places on the tree where one branch breaks into two or more branches. We shall also refer to the root and the ends of all of the branches as nodes.

Another idea which is key to the concept of a tree is that of ancestors and descendents. All nodes are descendents of the root node. Or, to look at it another way, the root node is the ancestor of all nodes. A direct descendent or ancestor is a node which is directly connected (by a branch) to another node. Referring to Figure 4, **motd** is a direct descendent of **etc.** The same relationship may be looked at as **etc** is a direct ancestor of

motd. The terms parent and child may be substituted for ancestor and descendent.

Please also refer to the section on trees in the Chapter Getting Started.

Cromix File Structure

The Cromix file structure is a tree whose nodes are composed of directories of ordinary files and other (descendent) directories. The highest level directory is called the **root** directory.

Types of Directory Entries

Directory — A directory appearing as an entry in another directory is a descendent directory.

File — A file appearing as an entry in a directory is an ordinary file.

Device

Character Device
A character device is a sequential access device (i.e., terminal, printer, etc.)

Block Device
A block device is a random access device which can maintain a file system (i.e., a disk).

. — A single period refers to the directory in which the entry occurs. Although it is never displayed, this entry is present in every directory. Note that the period is not a directory name but a reference to a directory. This reference always assumes the value of the current directory.

^ — A caret (up-arrow) refers to the ancestor directory. Although it is never displayed, this entry is present in every directory except the root directory. There is no ancestor for the root directory. Note that the caret is not a directory name but a reference to a directory. This reference always assumes the value of the ancestor directory.

Path name

A path name locates a file or directory within the file structure. The simplest form of a path name is a file name. If a file name is specified and no directory name is given, the file is assumed to be in the current directory.

All names within a path name must be separated by slashes (/). Each succeeding directory name and the final directory or file name must be a descendent of the previous directory.

If the path name is that of a directory, only directories will appear in the path name. If the path name is that of a file, the last item in the path name will be a file name.

A path name may contain a maximum of 128 characters. The first entry in a path name must be one of the following:

    /
    ^

    directory name
    file name

Notice that the caret (indicating an ancestor directory) may only appear as the first entry in a path name and may be followed by one or more additional carets. Each successive caret indicates another generation ancestor directory. Multiple carets are not separated by slashes. If an attempt is made to specify a directory which would be an ancestor of the root directory, the Cromix Operating System will proceed as though the root directory had been specified.

Absolute Path Name

An absolute path name locates a file or directory relative to the root directory. This type of path name always begins with a slash (/) to indicate the root directory and may be followed by any number of directory names.

The following examples refer to Figure 4 and make no assumptions about the current directory.

/cromix.sys	references the file cromix.sys located in the root directory

/etc/who — references the file who located in the directory etc which is located in the root directory.

/letters/personal/fred — references the file fred located in the directory personal which is located in the directory letters which is in turn located in the root directory.

Relative Path Name

A relative path name locates a file or directory relative to the current directory. The invisible directory entry indicating the ancestor directory (^) may be useful in the definition of a relative path name. Note that the slashes in a relative directory are used as delimiters and do not refer to the root directory.

The following examples refer to Figure 4 and assume the directory named **personal** is the current directory:

^/business — references the directory business located in the ancestor directory letters.

^/business/jones — references the file jones located in the directory business which is located in the (ancestor) directory letters.

rose/junel4 — references the file junel4 located in the descendent directory rose.

Assume the root directory is the current directory:

etc/motd — references the file motd which is located in the directory etc which is located in the root directory.

Cromix File Protection

The Cromemco Cromix Operating System offers protection for files on many levels.

All files may be opened for exclusive or non-exclusive access. A file which is opened for exclusive access may not be opened by another process until it has been closed by the process which opened it originally. If a file is opened for non-exclusive access, it may be opened and accessed by more than one process simultaneously.

There are four categories of access privileges:

execute
append
read
write

One or more of these access privileges may be assigned to the owner, a specified group of users, and all other users.

Execute indicates that the file may be executed.

Append indicates that data may be added to the end of the file. Data may be written to the file at a point past the end of file at which time the end of file indicator is moved to the end of the new data. If append access is specified, and no other type of access is permitted, the data which is appended may not be read.

Read indicates that the file may be read.

Write indicates that the file may be written to.

No one access privilege implies that any other access privilege is or is not granted. The categories of access privileges are normally combined to provide meaningful data handling. For example, a user having write access to a file normally will be provided with read access in addition.

Names

File and Directory Names

Any name within the Cromix Operating System (including file, directory, and device names) may contain from 1 through 24 characters from the following set:

A-Z, a-z, 0-9, $, _, and .

If a period is the first character of a name it will be an invisible name and will not normally be listed with the rest of the directory. Refer to the **list** utility, **-a** option.

The Cromix Shell, which processes commands from the console, does not distinguish between upper and lower case characters in file and directory names. On entry, all names are converted to and stored as null terminated strings of lower case characters.

Ambiguous File and Directory Names

The Cromix Shell will convert ambiguous file and directory names into a list of names which match the specified pattern. These names may be used by any program which is designed to accept a list of names.

The **asterisk (*)** will match any string of zero or more characters. For example:

```
a*b will match ab
               axb
               axyb
               a$$$b
```

A **double asterisk (**)** will match any name combined with any extension. In other words, it will match all names. For example:

```
** will match  all names
```

The **question mark (?)** will match any single character. For example:

```
a?b will match axb
               a1b
               a$b
```

Square brackets ([]) may be used to indicate that several single characters are to be substituted in the location of the brackets. For example:

```
a[xyza-d] will match
               ax
               ay
               az
               aa
               ab
               ac
               ad
```

A range of characters may be specified (as above) by the delimiters of the range separated by a hyphen (-).

File Naming Conventions

The Cromix Shell looks for three types of file name extensions and interprets these extensions as having special significance. (A file name extension is the portion of a file name which follows the final period embedded within the file name.)

A file name extension of **bin** indicates that the file is an executable file which will run directly under the Cromix Operating System.

A file name extension of **com** indicates that the file is an executable file which makes use of CDOS system calls. The Cromix Operating System will automatically load the CDOS Simulator with this type of file.

The file name extension **cmd** indicates that the file is a Cromix Shell program. The Shell will interpret each line of a **cmd** file as a Shell command line.

Refer to the section on Command Syntax for additional discussion on the use of file name extensions.

The /etc Directory

As Cromix is shipped, there are several files which have special significance. One group of important files is found in the directory **/etc.**

When listed, the contents of this directory appears as:

```
% l /etc<CR>

Directory: /etc
        144     1 account
         20     1 motd
        128     1 mtab
         76     1 passwd
         41     1 startup.cmd
         80     1 ttys
         48     1 who
```

Account

The **account** file may optionally be included in the **/etc** directory. When it is present, information concerning users logging on and off the system will be written to the account file.

Records in the account file are 48 bytes long. The first 16 bytes in each record indicate the terminal device on which the user was logged in. In a single user system this device will always be the console, while in a multi-user system this device will be tty1 through tty6. The next 16 bytes are the user name. Following this are three bytes representing the date, three bytes representing the time, two bytes containing the user id, and two bytes containing the group id. The last six bytes are reserved for future use. A plus sign (+) in the **login** user name field indicates when the system was booted.

The **who** utility may be used to display the account file. Please refer to the **who** utility for further information.

Motd

The **motd** file is the **message of the day** file. The contents of this file will be displayed each time a user logs on to the system. The user may edit this file to display any desired message. This is an informational file and contains no commands to the system.

Mtab

The **mtab** file contains the **mount table**. When the **mount** command is given with no arguments the mtab file is consulted and a list of mounted devices is displayed. This file must not be edited by the user.

The mtab file contains one 128 byte record for each disk which is mounted (on line). The first 32 bytes of each record contain the device name which is left justified and null padded. The last 96 bytes of each record contain the dummy path name where the device is mounted. The first record in mtab always specifies the root device.

Passwd

The **passwd** file contains information about each user. This information includes an encryption of any required password as well as restrictions on the user.

Each line of the passwd file represents one user. Each line has six fields which are separated by colons.

The first field is the user name. This is the name which must be typed in response to the Cromix Operating System prompt **login:**. The second field is an optional encrypted password. Refer to the passwd utility for information on adding, deleting or changing passwords.

The third and fourth fields are the user and group identification numbers. Each of these fields is an unsigned integer between 0 and 65535. A zero in the user field indicates a privileged user. A zero in the group field indicates that the user is not a member of any group. Any other number only has significance within a given system.

The fifth field is the initial directory. This will be the user's current directory immediately after logging on. The last field is an optional command line. If this line is blank, the user may run the Shell program. If any other command line appears here, execution of the command line will automatically begin when the user logs on, and the user will automatically be logged off when execution of the command line terminates.

Startup.cmd

The **startup.cmd** file is a file containing Shell commands which are to be executed when the system is started up. As shipped, this file contains commands to execute the date and time programs which are used to set the system clock.

Ttys

The **ttys** file contains a list of eight possible terminals and pertinent information for each terminal. This file must be edited using the Screen Editor in order to change the number of terminals which may be attached to the system.

Each line in this file represents one terminal. The first entry on each line is a one or zero. A one indicates that the terminal is present, a zero indicates that it is not.

The next column is delimited by a colon and represents the baud rate of the terminal. The baud rate for any one of the terminals may be one of the following: 19200, 9600, 4800, 2400, 1200, 300, 1500, 110, N, or A. **A** indicates that the baud rate will be automatically established when the user presses RETURN several times. **N** indicates no change in the baud rate. The first console must **not** be set to a baud rate of 19200.

The third column is delimited by a colon and has the name of the terminal. The terminals are named tty1 through tty8.

Who

The **who** file contains information on all users who are currently logged on the system. The format of the **who** file is identical to that of the account file.

The **who** utility may be used to display the contents of the **who** file. Please refer to the **who** utility for additional information.

Setting Up - Software

This section describes how to set up the Cromix Operating System software for users in addition to the privileged user. Please refer to the section titled Setting Up - Hardware for hardware considerations.

If the person who is going to set up the Cromix Operating System is not familiar with it, then it is recommended that the Getting Started section be read before proceeding.

As the Cromix Operating System is shipped there is one valid user name. That name is **system** and is the name of the privileged user. As shipped, there is no password associated with the user **system.** The privileged user is the only user who can establish other users on the Operating System.

The privileged user must first be logged in. This is accomplished by typing **system** in response to the **login** prompt.

Establishing a New User

A new user may be added by means of the **passwd** program. In the following session the user logs on as the privileged user **system** and then creates a new user **fred** with a secret password **mountain:**

```
Login: system<CR>

Logged in system Jun-24-1980 17:12:15 on console
# passwd -n<CR>

Name: fred<CR>
Password: rsyc5nzk
User number: 5<CR>
Group number: 0<CR>
Directory: /fred<CR>
Starting Program: <CR>

Name: <CR>
#
```

The **passwd** program first prompts for a user name. The response to this prompt is the user name which the user will type in response to the Login: prompt when logging in.

Next, the program prompts for the user password.

If no password is desired, type a **<CR>** in response to this prompt. Notice that the password is never displayed on the console. When a privileged user is entering a password while running the **passwd** program, the password encryption is displayed after the password and **<CR>** have been entered. When a user is logging on, nothing is displayed when the password is entered. In the above example, the password **mountain** was typed in response to the Password: prompt. The **passwd** program displayed the password encryption as rsyc5nzk.

The next two prompts are for the user and group identification numbers. Each of these fields is an unsigned integer between 0 and 65535. A zero in the user field indicates a privileged user. A zero in the group field indicates that the member is not a member of any group. Any other number only has significance within a given system.

The Directory: prompt allows the specification of an initial directory. This will be the user's current directory immediately after logging on. If this directory does not exist when the user logs on, the root directory will be the user's current directory.

Finally, the passwd program prompts for a Starting program:. If a **<CR>** is typed in response to this, the user will have full use of the Shell program. If the name of a program is entered here, the user will be brought up running the specified program and will be logged out upon exiting from the program. Any valid Shell command line may be entered in response to this prompt.

Deleting a User

A user may be deleted from the list of users (/etc/passwd file) by running the passwd program with the **-d** option. In the following example, the user **fred** who was established above, will be deleted:

```
# passwd -d<CR>

Name: fred<CR>

Name: <CR>
#
```

Note that only the privileged user may delete a user.

Changing the Password

When called without any options specified, the **passwd** program will allow the privileged user to change any user's password and will allow any user to change his or her own password. To change a password, call the passwd program as follows:

```
% passwd<CR>
Name: fred<CR>
Password: hjrft5zw

Name:<CR>
```

Notice that once again the password encryption is displayed only after the password and a **<CR>** have been entered.

Changing the User Characteristics

If the privileged user has occasion to change user characteristics other than the password, the user must be deleted and added again with the new characteristics specified.

Setting Up - Hardware

This section describes how to set up the Cromix Operating System hardware. Please refer to the section titled Setting Up - Software for software considerations.

Memory Boards

The Cromemco Cromix Operating System is designed to operate with from two to seven Cromemco 64KZ Random Access Memory Boards. One 64KZ RAM board resides in bank 0 and is reserved for the Operating System. This board also resides in all unused banks.

A minimum system requires two 64KZ RAM boards. One of these will be reserved for the Operating System and the other will be dedicated to the single user.

Each additional 64KZ RAM board allows an additional user to be logged on at the same time and/or an additional process to be executed concurrently.

Note that what is termed a **single user system** (2 memory boards) can support several users, but that **no two processes can be executed simultaneously.** A **Shell command** is **not** considered a process because it is executed within the system bank and does not require any user memory. Thus, a single user system can execute a Shell command **while** a process is being executed. In addition, a single user system can drive the printer through the system bank by means of the Spool utility. This allows a single user system to print a file and execute a program at the same time.

The number of users or, more accurately, the number of concurrent processes which a system will support can be determined by consulting the following table:

Amount of Memory vs. Number of Users

Number of 64KZ Boards	**Number of users**
2	1
3	2
4	3
5	4
6	5
7	6

The switches on the 64KZ memory boards may be properly set by referring to the following two pages of 64KZ switch settings.

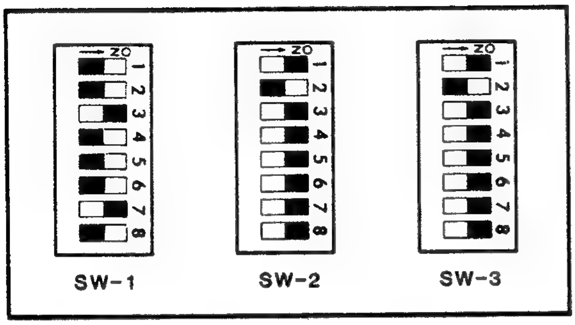
One board should be established as the system bank by setting the switches as indicated for the appropriate number of users. Refer to the diagrams entitled **64KZ System Bank Switch Settings. Only one board should be set according to the diagrams in this table.** The switches on additional 64KZ boards should be set in the manner described by the following paragraph.

The other board(s) should be established as user bank(s) by setting the switches as indicated in the diagrams entitled **64KZ User Bank(s) Switch Settings.** The switches on one board should be set as indicated by the diagram under the title **bank 1.** If there is a second board, it should be set according to the **bank 2** diagram. Additional banks should be established **in numerical order** for as many 64KZ boards as will be used.

If additional memory is added to the system at a later time, it is important to remember to change the switch settings on the system (bank 0) board.

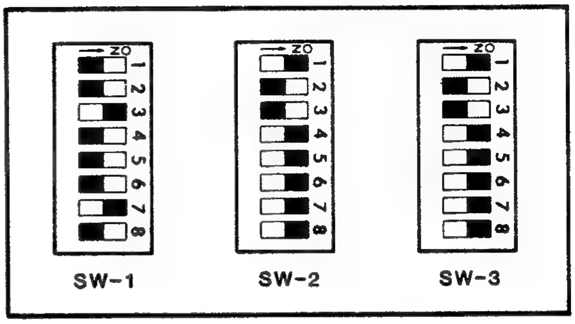
BANK 0 (1 user system)

64KZ board



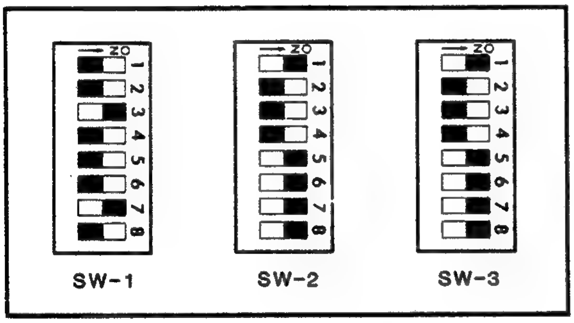
BANK 0 (2 user system)

64KZ board



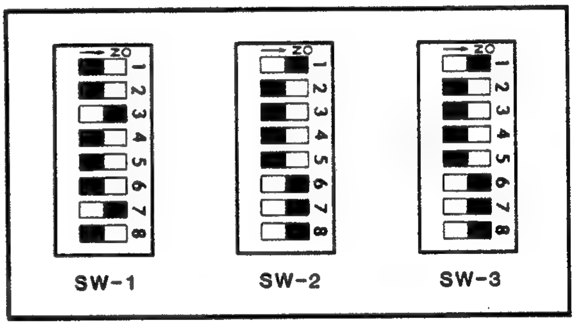
BANK 0 (3 user system)

64KZ board



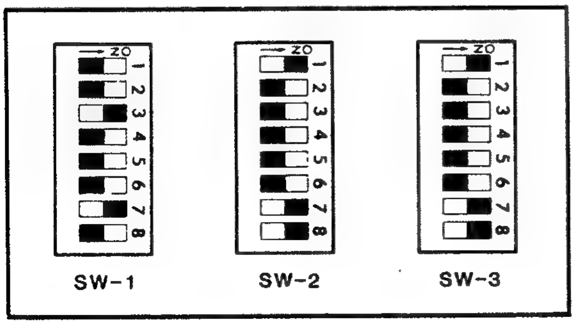
BANK 0 (4 user system)

64KZ board



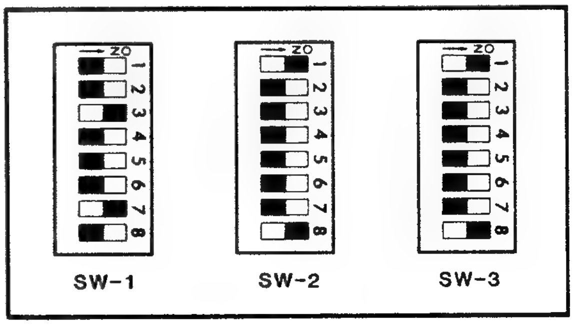
BANK 0 (5 user system)

64KZ board



BANK 0 (6 user system)

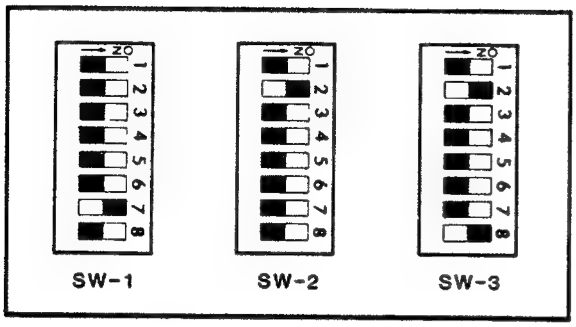
64KZ board



64KZ System Bank Switch Settings

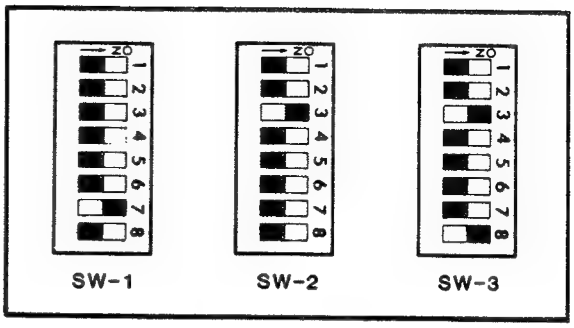
BANK 1

64KZ board



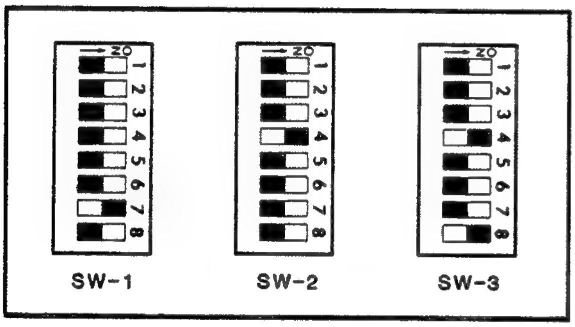
BANK 2

64KZ board



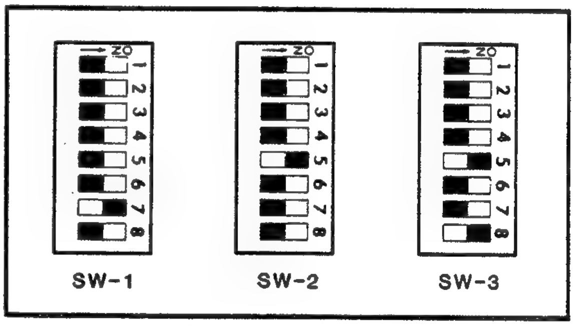
BANK 3

64KZ board



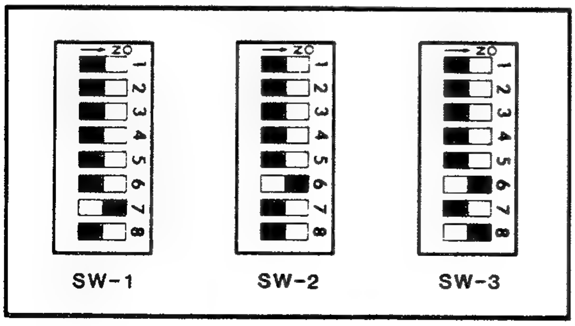
BANK 4

64KZ board



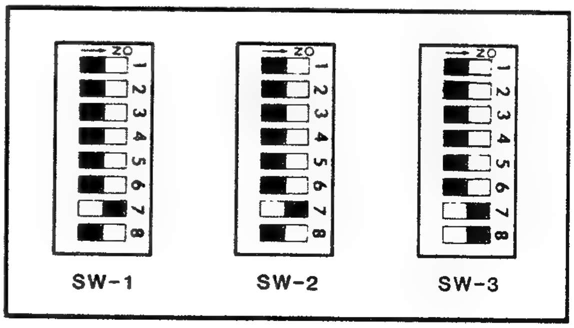
BANK 5

64KZ board



BANK 6

64KZ board



64KZ User Bank Switch Settings

Floppy Disk Controller

The following switch settings are recommended for the 4FDC or 16FDC disk controller. Note that switch sections 5 through 8 only apply to the 16FDC.

4FDC & 16FDC Switch Settings

Switch Section	Setting
1:	off
2:	on
3:	on
4:	off
5:	off
6:	off
7:	off
8:	off

Terminal Interface

The initial terminal is interfaced through the port provided for this purpose on the floppy disk controller board. A single user system does not require a TU-ART board.

Up to five additional terminals may be attached to the system by means of Cromemco TU-ART interface boards. Each TU-ART is capable of interfacing two terminals so that a maximum of three of these boards will be needed.

TU-ART Switch Settings

Switch Section	TU-ART #3	TU-ART #2	TU-ART #1
1:	off	off	off
2:	off	off	off
3:	on	on	on
4:	on	off	off
5:	on	off	on
6:	off	on	on
7:	off	off	off
8:	on	off	on
9:	on	off	off
10:	off	on	on

TU-ART #1 will service user(s) two and (if required) three through its serial ports A (port 20h) and B (port 50h), respectively.

If the system has more than three users, TU-ART #2 will service user(s) four and (if required) five through its serial ports A (port 60h) and B (port 70h), respectively.

If it is a six user system, TU-ART #3 will service user six through its serial port A (port 80h).

Printer Interface

The Cromemco PRI printer interface board will support one fully formed character printer such as the Cromemco 3355A and one dot matrix printer such as the Cromemco 3703.

The following table contains the recommended switch settings for the PRI board. Switch #1 is to the right (closest to the J1 connector) and affects the dot matrix printer. Switch #2 is to the left (closest to the J2 connector) and affects the fully formed character printer.

PRI Switch Settings

Switch Section*	Switch #1 Settings	Switch #2 Settings
1:	on	on
2:	on	off
3:	off	on
4:	off	off
5:	on	off
6:	off	off
7:	on	on
8:	on	on

* Refer to the switch section numbers on the switch itself and not to those on the printed circuit board legend.

Priority Interrupt Cable

The priority interrupt cable is a single wire with several connectors at regular intervals along its length. This cable must run between all of the following boards in the system: PRI, 4FDC or 16FDC, and all TU-ARTs. If the system has no PRI or TU-ART then no priority interrupt cable is required. The cable may run between these cards in any order but must go from the priority interrupt cable connector **out** pin on one board to the **in** pin on the next board, and so on.

The Cromix Shell

The Cromix Shell is the program which interprets and processes all commands as they are entered at the console.

The Shell insures that arguments which are typed on the command line are available for use by the called programs. It also allows more than one command to be entered on a line (sequential and detached processes) as well as allowing output to be sent to a file and input to come from a file (redirected I/O).

The Shell handles all file and device dependent information. All directories are created, changed, and displayed by using Shell commands.

The reader is referred also to the utility programs. This group of programs performs many functions which are similar to those performed by the Shell.

In this manual the term **command** will be used to refer to Shell commands which are intrinsic to the Cromix Operating System. The term **utility** will refer to utility programs which are stored on the disk. A command will execute within the system bank of memory while a utility is a program which requires, as does any program, additional memory for execution.

The Cromix Shell makes use of three standard files. These are the standard input file, **stdin,** the standard output file, **stdout,** and the standard error file, **stderr.** As shipped, when a user logs on, all three files refer to the console. That is, standard input for the Shell comes from the console keyboard and standard output to the Shell as well as error messages go to the console screen. Unless explicitly stated otherwise, the reader should assume that stdout, stdin, and stderr all refer to the console.

Command Syntax

The Cromix Shell assumes that each command has the following syntax:

```
filename arg1 arg2 ...
```

where **filename** is the name of a file and **arg1, arg2, ...** are optional arguments. The Shell program will search for **filename** as follows:

1. Current directory
   a. filename.bin
   b. filename.com
   c. filename.cmd

2. Default directory
   a. /bin/filename.bin
   b. /bin/filename.com
   c. /cmd/filename.cmd

If the file is not found in any of the above locations, an error message will be displayed. If the file is found, it will be treated in accordance with the file naming conventions outlined elsewhere. If a file has an extension of **cmd,** the command **Shell** is assumed to appear at the beginning of the command line.

Sequential and Detached Processes

More than one Shell command may appear on a single input line. A command which is followed by a semicolon (;) indicates that any following command should be executed only when the process which was initiated by the first command has finished execution. This is termed **sequential processing.**

A command which is followed by an ampersand (&) indicates that the process specified by the command should be executed as a **detached process** and that any subsequent command on the line should be executed as a **concurrent process.**

When a detached process begins execution, a process identification number (**PID**) is displayed on the terminal. One additional bank of memory is required for each additional detached process which is to be executed concurrently. If there is not enough memory in the system, an error message will be displayed.

For example, if **a** and **b** are commands which will each begin execution of single processes, then:

```
     % a;b
```

will cause process **a** to begin and complete execution before process **b** begins execution (sequential processing), and:

```
     % a&b
```

will cause process **a** to begin execution in the detached mode and process **b** to begin normal execution at the same time (concurrent processing).

If a single command is given on a line terminated with an ampersand (&), the process specified by the command will begin execution in the detached mode, a PID number will be assigned and displayed, and the Shell will immediately prompt for another command.

Execution of the Shell command **wait** will delay execution of any additional commands until all detached processes have finished execution.

Redirected Input and Output

Output which would normally go to the standard output device (the console) may be redirected to a file. This file may be an ordinary file or a device such as the printer. Redirection of output is accomplished by entering a greater-than sign (>) followed by the output file or device name on the command line.

```
     % ty novel.txt > xx
     % > xx ty novel.txt
```

Either of the above commands will execute **ty novel.txt** and send all output from the process to the file named **xx.** The redirection of output will create the named file if it does not exist, and will write over the contents of the file if it does exist.

If a double greater-than sign (>>) is used for redirecting output, the output will be appended to (added to the end of) the specified file. If the file does not exist, the append specification will function in the same manner as the simple redirection of output. For example:

```
    % ty 0830 >> notes
    % >> notes ty 0830
```

Either of the above command lines will cause the file **0830** to be appended to the file **notes.**

Input which would normally come from the standard input device (the console) may be redirected to come from a file. Redirection of input is accomplished by entering a less-than sign (<) followed by the input file or device name on the command line.

```
    % proc <infile
    % <infile proc
```

Either of the above commands will execute process **proc** and obtain all input from **infile.**

Important Note

It is not a good idea to redirect output to the printer on multi-user systems. Although a command such as the following:

```
    % ty novel.txt > /dev/prt
```

will send the contents of the file **novel.txt** to the printer, if two users or processes attempt to do this, the results are not predictable.

Instead, users should use the Spool utility to utilize the printer.

Parentheses on the Command Line

Parentheses may be used to group commands on the command line. They may be used to cause output from several processes which are executing sequentially to go to the same file:

```
    % (a;b) > xyz
```

The same output file would result from the command line:

```
    % a > xyz ; b >> xyz
```

The command line above would cause the output from process **a** to go to file **xyz** and the output from process **b** to be appended to the same file.

Parentheses can also be used to cause two or more sets of sequential processes to execute simultaneously as detached processes. The following command line will cause processes a, b, and c to execute in one bank of memory while processes d, e, and f are executing in another:

```
% (a;b;c) & (d;e;f)
```

If the user has sufficient memory, the command line above can be terminated with an ampersand which would cause both sets of processes to execute in the detached mode and the Shell to immediately prompt the user for another command.

Quotation Marks on the Command Line

Pairs of quotation marks (") or apostrophes (') may be used to enclose strings of special characters on the command line. For example, the following command line will display a greater than sign within a message:

```
% echo "this is a special character:  > right"
this is a special character:  > right
```

If the quotation marks are omitted, the output will be redirected to the file named **right.**

Quotation marks are also used with the cdoscopy utility to specify ambiguous **CDOS** files.

Argument Substitution

Arguments from the command line will be substituted in order into a command (cmd) file for each appearance of #1, #2, #3, etc. Assume that the command file named **test.cmd** contains:

```
ty #2 #1
```

If the command:

```
% test file_x file_y
```

is given, the first argument, **file_x**, will be substituted for **#1** in the command file and the second argument, **file_y**, will be substituted for **#2.**

Wait Command

The **wait** command will cause the Operating System to wait until all detached processes have finished execution. The following command line will cause processes **a** and **b** to execute concurrently and then, when both of these processes have finished, will cause processes **c** and **d** to execute concurrently:

```
% a&b; wait; c&d
```

Shell command:	**CREATE** or **CRE**
purpose:	This command creates a file.
summary:	cre file-list
arguments:	list of one or more file path names
options:	none

This command is used to create one or more files.

Shell command:	**DELETE** or **DEL**
purpose:	This command deletes a file or directory.
summary:	del file-list directory list
arguments:	one or more file path names
options:	-v verbose This option will display pertinent information as files are deleted.

The files and/or directories specified by the path names are deleted. They are no longer accessible and the space which they occupied is available for other use.

In order for a directory to be deleted it must not

1. contain any files,
2. be the current directory for any user, or
3. be the root directory of a device.

Shell command:	**DIRECTORY** or **D**
purpose:	This command displays or changes the current directory.
summary:	dir [dir name]
arguments:	optional directory path name
options:	none

When given without any arguments, the Directory command will display the current directory.

When given with a directory path name, the Directory command will cause the specified directory to become the current directory.

Shell command:	**EXIT** or **EX**
purpose:	This command exits from a Shell.
summary:	ex
arguments:	none
options:	none

This command is used to exit from a Shell. If there is no higher level Shell active, the Cromix Operating System will log the user off.

Shell command:	**HELP**
purpose:	This command assists the user.
summary:	help
arguments:	none
options:	none

The Help command will display all of the permitted Shell commands. The portion of each command which is displayed in upper case letters is the only portion of the command which needs to be entered by the user.

Shell command: **LIST**

purpose: This command lists directory or file information.

summary:
```
list [-abdi] file-list
             directory list
```

arguments: optional file or directory path name(s)

options:

-a All files are listed including invisible files (those files whose name begins with a period).

-b A brief listing which contains only names is displayed.

-d Information about the directory is displayed (instead of the contents of the directory).

-i Inode numbers are displayed for each file.

If no path name is specified, the contents of the current directory will be listed. If a directory path name is given, the contents of that directory will be listed. If a file path name is given, information about that file will be listed.

Note that there is a Shell command **list** and a utility program **l** which perform similar functions. The Shell command, as differentiated from the utility program, does not produce a list in alphabetical order and does not require a separate bank of memory for execution.

Shell command:	**MAKDIR** or **MAKD**
purpose:	This command makes a directory.
summary:	makdir dirl [... dirN]
arguments:	one or more directory path names
options:	none

The directories are created as specified by the path names.

Shell command: **PROMPT**

purpose: This command changes the prompt.

summary: prompt [char]

arguments: Char is the new character which the Cromix Operating System will use as a prompt. This must be a single character. If no character is specified, the prompt will be changed to the pound sign (#) for the privileged user or to the percent sign (%) for any other user.

Note that changing the prompt from a percent sign to a pound sign does not give a user the privileges of the privileged user.

options: none

Shell command:	**RENAME** or **REN**
purpose:	This command changes the name and/or directory of a file.
summary:	ren oldfile1 newfile1 [... oldfileN newfileN]
arguments:	one or more pairs of file path names - existing path name first, followed by the new path name
options:	none

The Rename command will change a file name and/or the directory in which it is located.

This command will not move a file from one device to another.

Shell command:	**SHELL** or **SH**
purpose:	This command creates a Shell process.
summary:	shell [cmd file]
arguments:	optional command file
options:	none

When given without an argument, the Shell command will create a Shell process. When given with the name of a file, the Shell command forces a specific command file to be used. This can be useful if there are two files in the current directory with the same name, one having a file name extension of **bin**, the other **cmd**. If just the name of the file is entered, the **bin** file will be executed. If the Shell command is given with the **cmd** file, the command file will be executed. In all other cases, the Shell command is implicit when the name of a command file is entered.

Refer to the section of this manual entitled The Cromix Shell for additional information.

Shell command:	**TYPE** or **TY**
purpose:	This command displays a file in ASCII.
summary:	ty file-list
arguments:	one or more file path names
options:	none

The Type command will display the file(s) specified by the path name(s). Type may be used only to display ASCII (text) files. The reader is referred to the Dump utility for information on displaying other types of files.

Shell command: **WAIT**

purpose: This command waits for all incomplete detached processes.

summary: wait

arguments: none

options: none

Execution of this command causes the Cromix Operating System to suspend operation until all detached processes which are currently being executed finish their execution.

UTILITY PROGRAMS

The Cromix utility programs perform many necessary functions. They are similar to and are used in conjunction with the Cromix Shell commands.

As contrasted to the Shell commands, the utility programs are not intrinsic to the Cromix Operating System but must be called off the disk when needed. Also, while Shell commands do not require memory in addition to the system bank of memory for execution, utility programs must have an additional bank of memory.

utility: **ACCESS**

purpose: This program changes the access privileges associated with a file.

summary: access rewa.rewa.rewa file-list

arguments: flags specifier string followed by one or more file or directory path names

The flags specifier string is composed of three parts separated by periods. The first part indicates owner permitted access, the second indicates group access, while the third indicates public access. Each part is composed of zero or more of the following flags given in any order:

+	add the specified flags
r	read access
e	execute access
w	write access
a	append access

options: none

Append access does not imply read access.

Execute access to a directory means that the user who has been granted the access privilege may use the directory in a path name.

Read access to a directory means that the user who has been granted the access privilege may list the directory.

utility:	**BOOT**
purpose:	This program loads an operating system into memory.
summary:	boot [filename]
arguments:	optional file name
options:	none

If boot is entered alone, the file **/cromix.sys** will be loaded and execution will begin. When used in this manner, the boot utility can be used to re-boot the Cromix Operating System.

If boot is followed by a file name, the file is assumed to have a file name extension of **sys.** If the user needs to boot **CDOS** from the Cromix Operating System, the file **CDOS.COM** can be copied (using the cdoscopy utility) to the root directory and renamed **CDOS.SYS.** From this point, the user types **boot cdos** to load CDOS and begin execution under this operating system.

Note that the boot utility may only be executed by a privileged user.

utility: **CDOSCOPY**

purpose: This program copies files to and from CDOS disks.

summary: cdoscopy [-blvw] devname filel ...

arguments: Cromix device name and the name(s) of the file(s) to be copied

options:

-w	write to CDOS
-v	verbose (display pertinent information as files are copied)
-b	binary file (1Ah not stripped from the end of the file)
-l	list CDOS directory contents

The cdoscopy program will copy files from the Cromemco Disk Operating System (CDOS) to the Cromemco Cromix Operating System and vice versa.

The Cromix Operating System **cannot** read CDOS disks. Programs which are to be executed and data which is to be read under the Cromix Operating System **must** be transferred from CDOS formatted disks to Cromix formatted disks before execution can begin.

Note that if a file path name is specified, CDOS will only consider the lowest level file name (the part of the path name following the rightmost slash).

If an ambiguous CDOS file reference is used, it must be enclosed in quotation marks.

The file named **/usr/lock** must be present in order to execute the cdoscopy program.

Examples:

```
% cdoscopy -v fda "*.Z80"
% cdoscopy -vw hdl **
% cdoscopy -l fdb
```

These examples assume that the disks in drive A (fda) and B (fdb) and the hard disk (drive F or hdl) are all formatted as CDOS disks. The first example will copy all of the CDOS files on drive A (Cromix Operating System designation **fda**) with the file name extension of **Z80** into the current directory. Because an ambiguous CDOS file reference was used, it was placed inside quotation marks.

The second example will write all of the files in the current directory to the CDOS hard disk designated as F (Cromix Operating System designation **hdl**). No quotation marks were used because the ambiguous file reference was a Cromix Operating System ambiguous file reference.

The final example will display the directory of the CDOS disk in drive B (fdb).

Refer to the Appendix for a list of device names.

utility:	**CHOWNER**
purpose:	This program changes the owner or group of one or more files.
summary:	chowner [-vg] ownername file-list
arguments:	name or number of the owner to whom ownership is to be transferred list of one or more file names
options:	-v verbose -g change group

Note that the chowner utility may only be executed by a privileged user.

utility:	**CMPASC**
purpose:	This program compares two ASCII (text) files.
summary:	cmpasc filel file2
arguments:	2 file names
options:	none

This program will compare two ASCII files and report on differences in length and content.

utility:	**COMPARE**
purpose:	This program compares two files.
summary:	compare file1 file2
arguments:	2 file names
options:	none

This program will compare two files and report on differences in length and content.

utility: **COPY**

purpose: This program copies file(s) from one directory into another.

summary:

```
copy [-fv] file-list dirname
     [-fv] srcfile destfile
```

arguments: a list of file names followed by a directory name or a source file followed by a destination file

options:

-f force delete
If this option is invoked, the copied file will overwrite another file with the same path name (if one exists). If this option is not invoked and another file exists with the destination path name, an error will be generated and the copy program will be aborted.

-v verbose
This option will display pertinent information as files are copied.

The copy program will copy one or more files into a directory. This program does not alter the source file(s).

utility: **CPTREE**

purpose: This program will copy a tree.

summary: cptree [-vf] source destination

arguments: source directory and destination directory

options: -f force delete
If this option is invoked the copied files will overwrite another file with the same path name (if one exists). If this option is not invoked and another file exists with the destination path name, an error will be generated.

-v verbose
This option will display pertinent information as files are copied.

This program will copy the source directory along with all descendent directories and files to the destination directory. All links which exist in the source directory will be preserved.

utility: **DATE**

purpose: This program displays or alters the date.

summary: date [month date year]

arguments: optional month, date, and year

options: none

If no arguments are given, the current date is returned. If the month, date, and year are specified, the Cromix Operating System date is reset. Refer to the setdate and getdate system calls for more information.

Note that if the argument **x** is given, the date utility will prompt the user for the date.

utility: **DCHECK**

purpose: This program verifies that the internal structure of the directories is correct.

summary: dcheck [-f] [devname]

arguments: optional device name

options: -f fix directory structure

This program should be run on unmounted file systems. If the file system you wish to fix is the root, then this program should be run with no other users or tasks running at the same time. If another task is writing to the disk, the results of dcheck may be incorrect.

If the -f option is used while another task or user is accessing the disk, the directory on the disk may be irreparably damaged.

Important note: Immediately after running dcheck with the -f option icheck should be run with the -s option. After both programs are run, the system must be rebooted. Refer to the boot utility for additional information. It is not necessary to re-boot if the -f option is not used.

Messages Returned by Dcheck

Cannot read super block
The super block cannot be read.

Out of memory
The disk contains too many inodes for dcheck to check. Make a new disk with fewer inodes and use the copytree utility program to transfer the contents of the disk to the new disk.

Cannot read inode xxxxx
A disk i/o error occured while trying to read the inode.

Inode xxxxx, error reading directory
A disk I/O error occured while trying to read a directory.

Inode xxxxx, cannot read inode
A disk i/o error occured while trying to read the inode.

Inode xxxxx, directory with more than 1 parent
A directory has more than one parent that are not in the same directory. Use the ncheck utility program to locate the names of the files, delete the names that have different parents, then run dcheck with the -f option.

Inode xxxxx, directory with wrong parent
This error indicates that the inode is pointing to the wrong parent. Use dcheck with the -f option to correct this error.

Inode xxxxx, bad link count xxxxx, should be xxxxx
There are a different number of names in directories pointing to this inode than the inode expects. Use dcheck with the -f option to correct this error.

Inode xxxxx, more than 255 links
There are more than 255 names for this inode. Use ncheck to find all the names, delete enough names to bring the total number of names to 255 or less, then run dcheck with the -f option.

Inode xxxxx, bad inode number in inode
Each inode contains its own inode number. This error indicates that the specified inode has the wrong number. Use dcheck with the -f option to correct this error.

Inode xxxxx, unallocated inode with xxx links
This inode is unallocated but it has names pointing to it. Use ncheck to find these names, then delete the names.

Inode xxxxx, allocated inode with 0 links
This inode is still allocated although there are no names for it. Use dcheck with the -f option to correct this error.

Inode xxxxx, bad directory entry count
This inode is a dirctory. The number of directory entries in the inode differs from the actual number of directories. Use dcheck with the -f option to correct this error.

utility:	**DUMP**
purpose:	This utility displays a file in hexadecimal.
summary:	dump [-b #] file-list [-e #] [-s #] [-o #]
arguments:	one or more file path names
options:	-b first byte to be dumped -e last byte to be dumped -s swath width -o offset to be added to all displayed addresses

The Dump program will display the file(s) specified by the path name(s). Dump will display any type of file. The file will be displayed in hexadecimal with an ASCII equivalent to the side of the dump. All numeric arguments to the dump utility are assumed to be decimal numbers unless they are followed by an **h** (for hexadecimal).

Example:

```
DUMP -b 1000h -e 5000h filename
```

This will dump the file filename starting with the 1000th (hex) byte and ending with the 5000th (hex) byte.

utility: **ECHO**

purpose: This program echos its arguments to the console.

summary: echo text

arguments: any text

options: none

utility: **FREE**

purpose: This program displays the amount of unused space remaining on one or more devices.

summary: free [devnamel ... devnameN]

arguments: an optional list of device names

options: none

The free program displays the amount of unused space remaining on the specified device(s). If no device list is specified, then the free space is displayed for all currently active devices.

utility: **ICHECK**

purpose: This program checks the integrity of one or more file systems.

summary: icheck [-s] [-b blk# ...] [devname ...]

arguments: an optional list of device names

options: -s salvage and recreate free list

-b display information about blocks

Icheck will verify the integrity of a file system. After a power failure or after resetting the computer it is a good idea to run icheck on all devices which were mounted at the time the problem occurred to make sure the file systems are in order.

If no device names are specified, icheck checks the integrity of all mounted devices. The list of mounted devices is obtained from the file **/etc/mtab.**

If no options are specified, icheck will produce a report on the file system but will not alter it. A sample report and explanation follow.

Important notes:

Immediately after running icheck with the -s option, the system must be re-booted. Refer to the boot utility for additional information. It is not necessary to re-boot if the -s option is not used.

Do not execute the icheck utility when other processes are being executed. This includes detached processes as well as other user processes.

```
% icheck<CR>

Device: /dev/hd0

Blocks missing:                        0
Bad free blocks:                       0
Duplicate blocks in free list:         0
Bad blocks:                            0
Duplicate blocks:                      0
Device files:                         16
Ordinary files:                      269
Directories:                          44
Blocks used in files:             13,546
Indirect blocks:                     172
Free blocks:                       6,212
Free inodes:                       3,871
```

Blocks missing

All disks (also referred to as block devices) are divided into units of allocation called **blocks.** A block is 512 bytes. Every block should appear in a file or in the **free list.** Blocks appearing in files include those which are permanently physically assigned as either system or inode blocks. The free list is a list of all blocks available for use.

A block is **missing** if it does not appear in a file or in the free list. Missing blocks do not compromise the integrity of the file system and the problem does not need to be corrected immediately. If a block is missing it is simply not available for use.

The problem may be corrected by executing icheck with the **-s** option.

Bad free blocks

This message pertains to blocks which are located in the free list. The term **bad** indicates that the block number is out of range. A block number can be out of range if it is:

1. past the end of the disk,
2. in the system area of the disk, or
3. in the inode area of the disk.

Bad free blocks **do** compromise the integrity of the file system and the problem should be corrected immediately by executing icheck with the **-s** option. No files will be affected.

Duplicate blocks in free list

This message indicates that the same block number appears twice in the free list.

Duplicate blocks in the free list **do** compromise the integrity of the file system and the problem should be corrected immediately by executing icheck with the **-s** option. No files will be affected.

Bad blocks

This is similar to **Bad free blocks** except that the Bad blocks appear in files.

Bad blocks **do** compromise the integrity of the file system and the problem should be corrected immediately, as follows.

Icheck will report the inode number of the bad blocks. The ncheck utility must be used to determine the names of the files containing the bad blocks and these files must be deleted.

Duplicate blocks

This is similar to **Duplicate blocks in free list** except that the Duplicate blocks appear in files.

Duplicate blocks **do** compromise the integrity of the file system and the problem should be corrected immediately, as follows.

Icheck will report the inode number of the duplicate blocks. The ncheck utility must be used to determine the names of the files containing the duplicate blocks and at least one of these files must be deleted. After this, the icheck utility should be run with the -s option.

Messages Returned by Icheck

Cannot read super block
The super block cannot be read.

Out of memory

The disk contains too many inodes for icheck to check. Make a new disk with fewer inodes and use the copytree utility program to transfer the contents of the disk to the new disk.

Cannot read inode xxxxx
A disk i/o error occured while trying to read the specified inode.

Not a block device: "device name"
The device specified is not a block device.

Inode xxxxxx, ---- Bad usage count ----
This inode has an incorrect usage count. The usage count is used by the usage utility program to calculate the amount of disk space used. This error can be corrected by running icheck with the -s option.

Inode xxxxxx, ---- Cannot write to inode ----
This error message occurs when icheck is attempting to correct an inode and an error occurs.

Block xxxxxx, inode xxxxxx, ---- block used in file ----
This is not an error message. This message is displayed when the -b option is used. It indicates the number of the inode in which the specified block is used.

Block xxxxxx, inode xxxxxx, ---- bad block number ----
Refer to the previous discussion of Bad blocks.

Block xxxxxx, inode xxxxxx, ---- duplicate block number ----
Refer to the previous discussion of Duplicate blocks.

Block xxxxxx, ---- block missing ----
This message is printed when the -b option is used to find the status of a certain block and the block is missing. Refer to the previous discussion of Blocks missing.

Block xxxxxx, ---- block in free list ----
This message is printed when the -b option is used to find the status of a certain block and the block is in the free list.

Block xxxxxx, ---- bad free block ----
Refer to the previous discussion of Bad free blocks.

Cannot write free list block xxxxxx
When running icheck with the -s option, the free

list is recreated. This error message is printed when there is an error in writing the free list.

Cannot read block xxxxxx
This message is printed when a block cannot be read.

utility: **IDUMP**

purpose: This program displays the contents of an inode.

summary: idump blockdev inode list

arguments: block device name
list of one or more inode numbers

options: none

This utility will display the contents of the specified inodes.

utility: **INIT**

purpose: This program initializes a disk.

summary: init

arguments: none

options: none

This utility will destroy all data on the specified disk.

The init program will prompt the user to determine which disk is to be initialized and what type of initialization is to be performed.

If a **RETURN** is entered in response to any of the questions followed by square brackets ([]), the init program will take the value specified within the brackets as the desired response. These default responses are standard and should normally be used.

After a disk is initialized, the make file system (makfs) utility program should be run on the newly initialized disk. After this, the disk may be mounted as a Cromix Operating System format disk.

utility: **L**

purpose: This program lists directory or file information.

summary: l [-abdeil] file-list

arguments: optional file or directory path name(s)

options:

-a All files are listed including invisible files (those files whose name begins with a period).

-b A brief listing which contains only names is displayed.

-d Information about the directory is displayed (instead of the contents of the directory).

-e Everything is displayed.

-i Inode number is displayed instead of file size.

-l A long list of information is displayed. This option does not display as much information as the everything option.

If no path name is specified, the contents of the current directory will be listed. If a directory path name is given then the contents of that directory will be listed. If a file path name is given then information about that file will be listed.

Note that there is a Shell command **list** and a utility program **l** which perform similar functions. The Shell command, as differentiated from the utility program, does not produce a list in alphabetical order and does not require a separate bank for execution.

utility: **MAIL**

purpose: This program handles mail between users.

summary: mail [user name]

arguments: optional user name(s)

options: none

When given without any arguments, the mail utility will display mail which has been sent to the user. After the mail has been displayed, the mail utility will ask if the user wants to save the mail. All saved mail will be appended to the file **/mbox.**

When given with one or more user names as arguments, the mail utility allows mail to be sent to one or more users. To do this, the message is entered after the **RETURN** is depressed at the end of the command line. A **CNTRL-Z** will terminate the message and return the user to the Cromix Operating System prompt.

If there is mail for a user, the user will be informed of this upon logging on the system.

utility: **MAKDEV**

purpose: This program makes a device file.

summary: makdev [-c] devname b/c devnum

arguments: device name

block or character device specification

device number

options: -c conditional
An error message will be displayed if there is no device driver corresponding to the device number specified.

The makdev utility program associates a device name with a device number and a device driver. After the execution of this program, all references to the specified device name will refer to the device specified by the device number.

utility: **MAKFS**

purpose: This program sets up the structure for a file system on a disk.

summary: makfs [-i #] devname

arguments: device name

options: -i number of inodes

Caution, this utility will destroy any existing data on the specified device.

This utility sets up the structure for a file system on a block device. This includes extablishing the number of inodes and the blocks which will be dedicated to the inodes, the blocks which will be dedicated to the system, and the blocks which will be dedicated to the user. The user is advised not to use the **-i** option.

Makfs must be run on all floppy disks and on some hard disks before mounting.

The makfs utility will warn and reprompt the user before destroying data.

utility: **MAKLINK**

purpose: This program makes a link to a file.

summary:

```
maklink    [-fv] file-list dirname
           [-fv] srcfile destfile
```

arguments: a list of file names followed by a directory name or a source file followed by a destination file

options:

-f force delete
If this option is invoked, the new link will overwrite another link with the same path name (if one exists). If this option is not invoked and another file exists with the link name, an error will be generated and the maklink program will be aborted.

-v verbose (display pertinent information as files are linked)

The maklink program will link one or more files into a directory. This program does not alter the source file.

utility: **MODE**

purpose: This program displays or alters the device mode.

summary: mode [devname] [option1 ... optionN]

arguments: an optional device name followed by an optional list of options

options: none

The mode program is patterned on the getmode and setmode system calls. Please refer to these for more information.

If no device is specified, the device from which the mode utility was called is assumed to be device in question. This method will normally default to the terminal calling the mode utility. However, if the mode utility is called from a command file, the disk drive upon which the command file is stored will be considered to be the source device. In summary, if the mode of the terminal is to be changed by a command file, the device must be explicitly specified.

If no options are specified, all options are displayed (getmode). Specified options are altered (setmode). The user may change any of the displayed options by entering **mode** followed by the option. Some option have values which may be changed (width, length, bmargin, etc.) while others can be turned on or off (pause, wrap, etc.). The value of the option is changed by following it with the desired value. If the option is preceded by a minus sign, it will be turned off. If option is preceded by a blank, it will be turned on. It is only necessary to type the portion of the option which is displayed in upper case letters.

Example:

The following call to mode will turn off the pause feature of the terminal:

```
% mode -pa
```

After this call, output to the terminal will not pause and wait for the user to type CNTRL-Q after displaying each set of 24 lines.

utility: **MOUNT**

purpose: This program enables access to a file system.

summary: mount [[-r] devname dummyname]

arguments: optional device name and a dummy file path name

options: -r read only

Refer to the mount system call for more information. When given without any arguments, mount lists the currently mounted devices.

Example:

```
% create newfilesys
% mount fdb newfilesys
% l
    .
    .
    .
145 D newfilesys

%
```

In the above example the programmer first created a dummy file. After mounting, the name of this dummy file will become the directory name of the root directory of the file system which is mounted. After unmounting this name will revert to being a dummy file name.

The mount command was given with the device name (refer to the appendix for a complete list of device names) of the location of the file system.

The list command shows that the new file system has been mounted and gives the name of the root directoy.

A file system which has been mounted **must be unmounted** by use of the unmount utility **before the mounted disk is removed from the system or the system is powered-down.** If this is not done the integrety of the data on the mounted system cannot be assured.

utility: **MOVE**

purpose: This program moves file(s) from one directory into another.

summary:

```
move [-fv] file-list dirname
     [-fv] srcfile destfile
```

arguments: a list of file names followed by a directory name or a source file followed by a destination file

options:

-f force delete
If this option is invoked, the moved file will overwrite another file with the same path name (if one exists). If this option is not invoked and another file exists with the destination path name, an error will be generated and the move program will be aborted.

-v verbose (display pertinent information as files are moved)

The move program will move one or more files from one directory to another directory. This program destroys the source file(s).

utility: **MSG**

purpose: This program allows messages to be sent between users.

summary: msg [-ny2] [user or dev name]

arguments: text terminated by CNTRL-Z

options:

-n disable incoming messages

-y enable incoming messages

-2 send message to the status line of a Cromemco 3102 terminal

The msg utility allows messages to be sent between users or from a user to a device. Sending a message to a device could be useful if a terminal was on line but no user was currently logged on.

If **msg** is typed and followed immediately by a **<CR>** then a message will be displayed which will inform the user of the status of incoming messages. Incoming messages may be disabled by the use of the **-n** and **-y** options.

When **msg** is followed by (optionally the **-2** option and) a user or device name and a **<CR>**, a message may be entered. The message will be transmitted to the destination user after each **<CR>** is depressed. A **CNTRL-Z** will terminate the message and return the originating user to the shell.

Two way communication may be established by the msg utility. When one user receives a message:

```
Message from xxxx
```

that user should type:

```
msg xxxx<CR>
```

This will allow both users to send messages to each other. In the above example, **xxxx** represents a user name.

If two way communication is established, it is recommended that a protocol be established to prevent the confusion which can arise when two messages are transmitted simultaneously. One suggested protocol follows. One user transmits at a time. A single **o** (short for over) is transmitted as the only character on a line to indicate the end of the message. Upon seeing the **o**, the other user responds, terminating the message with an **o**. When the entire communication is finished, one user should transmit **oo** (short for over and out) followed by a **CNTRL-Z**. The other user should type a **CNTRL-Z** also.

utility:	**NCHECK**
purpose:	This program displays file information.
summary:	ncheck [-i # # ...] [dirname]
arguments:	directory path name
options:	-i displays information about specified inodes only

The ncheck program displays the inode number, link count, and path name of all files contained in the specified directory and all subdirectories. If no arguments are supplied, ncheck uses the / directory.

utility: **PASSWD**

purpose: This program can be used to change a user password, add a user, delete a user, or change the group to which a user belongs.

summary: passwd {-dgn} [user1 user2...]

arguments: user1 user2...

options:

d This option deletes the specified user(s).

g This option is used to change the group to which a user belongs.

n This option is used to add new user(s).

The passwd utility has three functions. It may be used by any user to change that user's <u>own</u> password. It may be used by a privileged user to add and delete users from the list of users who may log onto the system. By using the delete function followed by the add function, the privileged user may change the log on status of any user.

In any one of the three modes of operation, the user name(s) may be specified either on the command line or during the execution of the passwd program.

If only the password is to be changed, enter the command **passwd** followed by a **RETURN.** The passwd program will then prompt for a user name and a new password.

utility: **PRIV**

purpose: This program allows any user to have the status of a privileged user.

summary: priv

arguments: none

options: none

The priv utility examines the **/etc/passwd** file for a user named **system.** If this user is not found, an error message is displayed and execution of the utility is aborted. If the user named **system** is found and if there is a password associated with the user, the priv utility prompts for the password. If the user responds with the correct password or if there is no password associated with the user **system,** a new shell is formed in which the user has the status of a privileged user. Upon exiting from the newly created shell, the user's previous status is reinstated.

utility: **PSTAT**

purpose: This program displays the status of a process.

summary: pstat [-abl]

arguments: none

options:

- -a All processes are listed. If this option is not selected, only those processes with the id of the user initiating the pstat call will be displayed.
- -b brief display
- -l long display

The pstat utility displays the following information on the status of a process:

PID	process identification number
state	state of process: Sleeping Ready Terminated
user id #	
group id #	
event (hex)	
bank	memory bank in which the process resides
command line	command line initiating the process

utility: **SCREEN**

purpose: This program allows the user to edit files.

summary: screen file name

arguments: name of file to be edited

options: none

Please refer to the Cromemco Screen Editor Instruction Manual (part number 023-0081) for a complete discussion of the Screen Editor. This write-up only covers those features of the Cromemco Cromix Screen Editor which are different from the Cromemco CDOS Screen Editor.

The Cromix Screen Editor is a special version of the Screen Editor which was designed to take advantage of some of the features of the Cromix Operating System. It utilizes Cromix Operating System calls and does not use the CDOS Simulator.

The only outwardly apparent difference to the user is the addition of the % command. This command will cause the Screen Editor to create a Shell process which will allow the user to execute any commands, provided there is enough memory in the system. Even without any additional memory, any of the Shell commands, such as **list** and **type**, may be used. The user can return to the Screen program at any time by entering the **exit** command in response to the Shell prompt.

utility: **SIM**

purpose: This program allows CDOS programs to run under the Cromix Operating System.

summary: none

arguments: none

options: none

Sim allows CDOS programs to run under the Cromix Operating System. The CDOS simulator is automatically loaded when a file with the file name extension **COM** is executed.

utility: **SPOOL**

purpose: This program causes files to be queued and sent to a printer.

summary: spool [-cdklpv] [pri] [devname] file-list

arguments: priority number

If no priority number is assigned, a value of five is assigned to the printing job.

device name

If no device name is specified, output is directed to **/dev/prt.** Two printers (one dot matrix and one fully formed character printer) may be attached to the system at once. With this configuration, the device name may be used to direct the output of the spool program to either one of the printers by using the device names **lpt1** and **typ1.**

file names or sequence numbers

File names must be used to add files to the printing queue. File names or the sequence numbers assigned by the spool program may be used to reference printing jobs for priority change or deletion.

options: Options which are used when adding files to the printing queue:

No option followed by a list of one or more files will add the specified files to the printing queue. A device name may be specified.

-d delete and enter

All specified files will be added to the Spool queue and deleted from the directory in which they reside. This option may include a device name. It must include a list of one or more file names.

-p priority
This option will assign a priority number to a printing job at the time it is initiated. The option must be followed by the desired priority number and may include a device name.

-v verbose
This option will cause a list of files being processed by the command to be displayed on the terminal as each file is processed. It may be used with all options except list and message.

<CR> message
This option allows the user to place a message in the printing queue. To accomplish this, type the program name **Spool** followed immediately by a **RETURN.** Follow this by the desired message terminated by **CNTRL-Z.** The file name of this message will be ----. This option may include a device name.

Options which are used to list the names of the files in the printing queue:

-l list
All printing jobs which are in the printing queue and which the user has initiated are listed in a table with the following information:

1. **Filename** of print file,
2. name of **User** requesting printing job,
3. **Sequence** number of printing job,
4. destination **Device** of printing job,
5. **Priority** of printing job,
6. **Pages** in printing job, and
7. **Lines** in printing job.

A privileged user always gets a list of all jobs in the printing queue.

-la list all
All printing jobs are listed in a table. Refer to the list option.

Option used to change the priority of file(s) in the printing queue:

-c change priority
All specified files which are in the Spool queue will have their priority set to the specified value. This option must be followed by a priority number. It must include a list of one or more file names or sequence numbers.

Option used to delete files from the printing queue:

-k kill
All specified files which are in the Spool queue will be deleted from the queue. If a specified file is currently printing, the printing will be aborted. This option must include a list of one or more file names or sequence numbers.

Purpose

The Spool utility program allows one or more users to send printing jobs to one or more printers in an orderly sequence which may be changed at any time.

Memory Requirements

When the Cromix Spool utility is called and requested to add files to the printing queue, the files are copied into a directory named **/usr/spool.** The execution of the Spool utility requires one bank of user memory, as does the execution of any utility program.

After the execution of the Spool program with any of its various options, the specified files will be sent to the printer **without the use of any user memory.** This is accomplished by a function which is intrinsic to the Cromix Operating System.

Other Requirements

The Spool utility will not function if the **/usr/spool** and **/usr/lock** directories are not present. If it is necessary to do so, these directories can be created by typing the following commands in response to the Cromix Operating System Prompt:

```
% makdir /usr/spool<CR>
% makdir /usr/lock<CR>
```

Destination Devices

Output from the Spool program may be directed to any character device which is located in the device table (**/dev**).

If no device is specified, **/dev/prt** is assumed.

Printing Queue & Sequence Numbers

As requests are made to print additional files, the Spool program forms a **printing queue.** Each file entered into the printing queue is given a unique **sequence number.** Once in the printing queue, files may be referenced by their file name or sequence number.

If two or more files in the queue have the same file name, a reference to that file name will refer to all files with the same name. For example, if the **k** (kill) option is used with a file name which appears more than once in the queue, all files with that name will be deleted from the queue. The sequence number can always be used to refer to a specific file.

Priority

Each file which is added to the printing queue is assigned a priority number in the range from 0 to 9. Zero is the highest priority and is reserved for a privileged user. If no priority is specified, a value of five is automatically assigned. A priority number must be specified when using the change priority option.

If two users request each request a print job with the same priority, the requests will be serviced on a first come, first served basis.

User Access

A user other than a privileged user only has access to files which that user placed in the printing queue. The priority of a file in the printing queue can only be changed by the user who initiated the printing request or by a privileged user. In a similar manner, only the privileged user or the user who added a file to the printing queue can delete the file from the queue (by use of the kill option). Any user can list **all** of the files in the printing queue by using the **la** (list all) option.

Ambiguous File References

Ambiguous file references must be used with caution.

When an ambiguous file reference is expanded, it generates a list of file names which match files in the current directory. An ambiguous file reference will work properly when giving the Spool program names of files to add to the printing queue.

An ambiguous file reference will not work properly when killing or changing the priority of files in the printing queue if files of the same name do not exist in the current directory. This would be the case if the **delete** option was used when the files were added to the printing queue or if the current directory had been changed by the user.

Examples:

For the following examples assume that the print files t, u, w, x, y, and z exist in the current directory. First, let us place each of these files in the printing queue:

```
% spool -v t u w x y z<CR>
t
u
w
x
y
z
%
```

Because the verbose option was used, the Spool program listed each of the files as it was copied to the spool directory. Next the list option will be used to display the printing queue:

```
% spool -l<CR>
   Filename        User       Seq   Device   Pri  Pages Lines
-> t               fred        36   29 prt   5        2    95
   u               fred        37   29 prt   5        2   107
   w               fred        38   29 prt   5        1    42
   x               fred        39   29 prt   5        2   115
   y               fred        40   29 prt   5        2   115
   z               fred        41   29 prt   5        3   160
```

The arrow at the upper left of the listing indicates that the file **t** is currently being printed. All of the jobs have a priority of 5 because no priority was indicated when the jobs were put in the queue.

Next we will change the priority of file **y** to 2 and change the priority of the file with the sequence number 39 (file **x**) to 3. Then we will delete the file u from the queue by use of the **k** option. Finally, we will add a message to the printing queue by use of the message option and display the revised printing queue.

```
% spool -c 2 y<CR>
% spool -c 3 39<CR>
% spool -k u<CR>
% spool<CR>
this is a message<CR>
<CR>
^Z% spool -l<CR>
   Filename        User      Seq   Device   Pri  Pages  Lines
-> t               system     36   29 prt    5      2     95
   y               system     40   29 prt    2      2    115
   x               system     39   29 prt    3      2    115
   w               system     38   29 prt    5      1     42
   z               system     41   29 prt    5      3    160
   ----            system     42   29 prt    5      1      2
```

Remember that a message must be terminated by a **CNTRL-Z** which will echo to the console as **^Z.**

utility: **TIME**

purpose: This program displays or alters the time.

summary: time [hour minute second]

arguments: optional hours, minutes, and seconds

options: none

If no arguments are given, the current time is returned. If the hours, minutes, and seconds are specified, the Cromix system clock is reset. Refer to the settime and gettime system calls for more information.

If the time utility is called with the argument **x**, the user will be prompted for the time.

utility:	**UNMOUNT**
purpose:	This program disables access to a file system.
summary:	unmount devname
arguments:	device name
options:	none

Refer to the unmount system call for more information.

A file system which has been mounted **must be unmounted** by use of the unmount utility **before the mounted disk is removed from the system or the system is powered-down.** If this is not done the integrety of the data on the mounted system cannot be assured.

utility:	**USAGE**
purpose:	This program displays directory size infomation.
summary:	usage file-list
arguments:	directory or file path name(s)
options:	none

The usage utility will display the physical disk space in blocks and the logical file space in bytes which is occupied by a directory and all of its descendent directories and files. If only a single file is specified, then the size of that file will be reported. If no path name is given, the root directory will be assumed.

Knowing the number of blocks occupied by a directory is useful when using the copy tree (cptree) utility.

utility: **VERSION**

purpose: This program displays the version number of the Cromix Operating System or a utility program.

summary: version [file path name of program]

arguments: optional utility name

options: none

When called without any argument, the version utility will display the version of the Cromix Operating System which is being run. When called with the name of a utility program, version will display the version number of that utility.

When called with the argument **/bin/****, the version numbers of all of the utilities in the **/bin** directory will be displayed. If there are a lot of utilites in the **/bin** directory, it may be necessary to break the list into two parts:

```
version /bin/[a-m]** ; version /bin/[n-z]**
```

This will produce an alphabetical listing of all of the utility programs with their version numbers.

utility: **WBOOT**

purpose: This program initializes the boot track of a floppy disk.

summary: wboot devname [BOOT PROGRAM]

arguments: device name

options: none

The wboot utility will write the contents of the **/etc/fdboot** (large floppy disk) or **/etc/sfdboot** (small floppy disk) file to the boot track of the disk in the specified device. This, together with the **cromix.sys** file in the highest level directory of the same disk, will allow this disk to be used to boot the Cromix Operating System.

utility: **WHO**

purpose: This program displays a list of users who are currently logged in.

summary: who [am i]

arguments: optionally: /etc/account

options: none

When the who utility is called without any arguments, the **/etc/who** file is examined and a report is displayed showing the users who are currently logged on together with the time each one logged on.

When followed by **am i,** the name of the user calling the who utility is displayed.

If the file **/etc/account** exists and the who utility is called followed by this path name, the information contained in the account file will be displayed.

SYSTEM CALLS

Calls to the Cromix Operating System are formed using a Z-80 restart instruction (RST 8) followed by a byte specifying the system call number.

The Cromemco Macro Assembler (version 03.07 and higher) now contains an opcode (JSYS) which forms these two bytes in the object code. JSYS takes one operand which is the Cromix System call number. For increased ease of reading and debugging code and for the convenience of the system programmer a file named JSYSEQU.Z80 is provided to facilitate system calls. This file contains EQUates for all of the system call numbers so that the calls may be made by name and the numbers do not have to be remembered.

For example:

```
     jsys .create    ;system call to create
                     ;and open a file

     jsys .close     ;system call to close
                     ;a file
```

All system calls require the specified calling parameters. In addition, some calls return parameters. Parameters are passed to and returned from system calls in registers or register pairs. All registers not specified as containing a returned parameter will be preserved through a system call.

If an error occurs during a system call, the carry flag (carry bit in the flag register) will be set and the a register will contain the error code.

system call: **.CACCESS**
number: 27h

purpose: This call tests channel access.

summary:

```
b    =   channel
jsys   .caccess
c    =   access bits
```

calling parameters: b — The b register contains the number of the channel whose access is to be tested.

return parameters: c — The c register contains the applicable access bits set:

AC.READ	read
AC.EXEC	execute
AC.WRIT	write
AC.APND	append

possible errors: ?filaccess

```
system call:   .CCHSTAT
     number:   23h

    purpose:   This call changes access privileges
               for a channel.

    summary:   hl  ->  path name
               c   =   owner
               d   =   group
               e   =   other
               jsys   .cchstat

    calling
 parameters:   hl   The hl register pair points to
                    the path name of the file.

               c    The c register contains the
                    status type which is to be
                    changed.

               d    The d register contains the new
                    value of the specified status
                    type.

               e    The e register contains a mask
                    of the status bits which are to
                    be changed.

                         AC.READ   read
                         AC.EXEC   execute
                         AC.WRIT   write
                         AC.APND   append
     return
 parameters:   none

   possible
     errors:   ?filaccess
               ?priv
```

Chaccess allowes the owner of a file to change file access privileges for the owner, group or others. Possible methods of access are:

read
write
append
execute
no access permitted

Any of these access methods may be combined. Please refer to the following page for a table of Cchstat calls.

Table of Cchstat Calls

Who*	C Register	Status Type	Location of New Information
p	ST.OWNER	owner	de = new value
p	ST.GROUP	group	de = new value
p&o	ST.AOWNER	access owner	d = new value, e = mask
p&o	ST.AGROUP	access group	d = new value, e = mask
p&o	ST.AOTHER	access other	d = new value, e = mask
p	ST.TCREATE	time created	de -> 6 byte buffer
p	ST.TMODIFY	time last modified	de -> 6 byte buffer
p	ST.TACCESS	time last accessed	de -> 6 byte buffer
p	ST.TDUMPED	time last dumped	de -> 6 byte buffer

*p = privileged user
o = owner

system call: **.CHDUP**
number: 0Ah

purpose: This call duplicates a channel.

summary:

```
b     =   existing channel
jsys    .chdup
c     =   duplicate channel
```

calling parameters: b — The b register contains the existing channel number.

return parameters: c — The c register contains the duplicate channel number which the system has assigned.

possible errors: ?notopen

system call:	**.CHKDEV**
number:	07h
purpose:	This call will check for the presence of a device driver.

summary:	d = type of device
	e = device number
	jsys .chkdev

calling parameters:	d The d register indicates the type of device: IS.BLOCK block device IS.CHAR character device e is the device number
return parameters:	none
possible errors:	?nodevice

system call: **.CLINK**
number: 25h

purpose: This call establishes a link to an open file.

summary:

```
b      =    channel
de     ->   new path name
jsys      .clink
```

calling parameters:

b The b register contains the channel number of the open file.

de The de register pair points to the file path name to be established (new).

return parameters: none

possible errors:

?badname
?isdir
?numlinks
?diraccess

system call:	**.CLOSE**
number:	0Bh
purpose:	This call closes a file.
summary:	b = channel jsys .close
calling parameters:	b The b register contains the channel number of the open file.
return parameters:	none
possible errors:	?notopen

Close empties all buffers associated with the specified channel number and disassociates the channel number from the file to which it was assigned.

```
system call:   .CREATE
     number:   08h

    purpose:   This call creates and opens a new
               file.

    summary:   hl  ->  path name
               c   =   access mode
               d   =   exclusive mode
               jsys   .create
               b   =   channel


    calling
 parameters:   hl   The hl register pair points to
                    a buffer containing the path
                    name of the file which is to be
                    created and opened.

               c    The c register contains the
                    access mode for opening the
                    file.  The following labels
                    represent the values of the c
                    register required in order to
                    establish the desired access
                    mode.

                       non-exclusive access
                         OP.READ   read only
                         OP.WRITE  write only
                         OP.RDWR   read/write
                         OP.APPEND append

                       exclusive access
                         OP.XREAD  read only
                         OP.XWRITE write only
                         OP.XWRITE read/write
                         OP.XAPPEND  append

                       truncate flag
                         OP.TRUNCF delete existing
                                   data

                       conditional flag
                         OP.CONDF  return error
                                   if file exists

               d    The d register contains the
                    mask for exclusive access.
                    Each of the specified bits must
                    be set to prevent the file from
                    being opened by more than one
                    user with the specified access.
```

```
                        The bits may be anded together
                        in order to set more than one
                        bit.

                           exclusive access bits
                             ^OP.READ    read only
                             ^OP.WRITE   write only
                             ^OP.RDWR    read/write
                             ^OP.APPEND       append

     return
 parameters:      b     The b register contains the
                        channel number which the system
                        has assigned to the file.

   possible
     errors:      ?filtable
                  ?badname
                  ?diraccess
                  ?isdir
```

Create will attempt to create a file with the specified path name.

If the file does not exist at the time of the system call it will be created and opened for the requested access.

If the file does exist and the conditional flag is set an error will be returned. If the file does exist and the conditional flag is reset the file will be opened.

If the file exists and is opened (as specified by the conditional flag) the existing data will be kept if the truncate flag is reset and will be discarded (the file will be truncated) if the truncate flag is set. An existing file may only be truncated if the user has write access privilege.

The channel number which The Cromix Operating System returns must be used for subsequent access to the file.

The created file will have the default access privileges. These are read and execute for group and others, and read, execute, write, and append for the owner.

system call:	**.CSTAT**
number:	21h
purpose:	This call determines the status of the file which is opened.

summary:

```
b    =   channel
c    =   desired information
jsys   .cstat
de   =   return value
hl   =   return value
```

calling parameters:

b — The b register contains the channel number of the file.

c — The c register contains the request to the system for the desired information. Refer to the table below.

return parameters:

dehl — The de (and in some cases the hl) register pair contains the requested information. Refer to the table below.

possible errors: ?badname

Cstat returns channel status information. Please refer to the table of Ctat calls on the following page.

Table of Cstat Calls

C Register	Information Returned	Location of Infomation
ST.ALL	all of inode	de -> 128 byte buffer
ST.OWNER	owner	de
ST.GROUP	group	de
ST.AOWNER	access owner	d
ST.AGROUP	access group	d
ST.AOTHER	access other	d
ST.FTYPE	file type	d = IS.ORDIN IS.DIRECT IS.CHAR IS.BLOCK e = device number
ST.SIZE	file size	dehl
ST.NLINKS	number of links	de
ST.INUM	inode number	de
ST.DEVICE	device number	d
ST.TCREATE	time created	de -> 6 byte buffer
ST.TMODIFY	time last modified	de -> 6 byte buffer
ST.TACCESS	time last accessed	de -> 6 byte buffer
ST.TDUMPED	time last dumped	de -> 6 byte buffer

system call:	**.DELETE**
number:	06h
purpose:	This call deletes a directory or directory entry.
summary:	hl -> path name jsys .delete
calling parameters:	hl The hl register pair points to a buffer containing the path name of the directory or file which is to be deleted.
return parameters:	none
possible errors:	?diraccess ?notexist

Delete will attempt to remove the specified directory entry. If the removed directory entry is the last link to the file then the file itself will be deleted and the space occupied by the file released. The contents of the file will be lost.

Write access is required for the directory containing the entry to be deleted.

If the file was open in a process at the time the system call was made and the specified directory entry was the last link to the file the directory entry will be deleted immediately. The file itself, however, will not be deleted until the active process closes the file.

In order for a directory to be deleted it must not

1. contain any files,
2. be the current directory for any user, or
3. be the root directory of a device.

system call:	**.DIVD**
number:	54h
purpose:	divide

summary:	dehl= dividend
	bc = divisor
	jsys .divd
	hl = quotient
	de = remainder

calling parameters:	
return parameters:	
possible errors:	?ovflo

```
system call:   .ERROR
     number:   1Ch

    purpose:   display error message

    summary:   dehl=   dividend
               a    =  same
               b    =  channel
               dehl=   same
               jsys   .error
```

calling parameters:

a — The a register shall remain as it was returned by the system call which generated the error.

b — The b register contains the channel number. This channel will receive the error message and is usually set to **stderr.**

dehl — The dehl register pair shall remain as it was returned by the system call which generated the error.

return parameters:

possible errors:

Error sends an error message to the channel specified by the **b** register.

The error system call should only be called immediatly after a system call which generated an error (if the carry bit in the flag register has been set).

Notice that errors may occur during calls to error and that this will set the carry bit.

```
system call:   .EXEC
     number:   4Ch

    purpose:   This call executes a program.

    summary:   de  ->  argument list
               hl  ->  path name
               jsys   .exec


    calling
 parameters:   de   The de register pair points to
                    a list of pointers.  The list
                    of pointers is terminated by a
                    null pointer (=0).  Each of the
                    pointers points to a null
                    terminated character string.
                    Each of the strings is an
                    argument which is passed to the
                    new program.

               hl   The hl register pair points to
                    the file path name.

     return
 parameters:   none (does not return)

   possible
     errors:   ?badname
               ?filaccess
```

Exec overlays the current process with the specified file and begins execution.

system call:	**.EXIT**
number:	46h
purpose:	This call exits from a process.
summary:	hl = termination status jsys .exit
calling parameters:	hl The hl register pair contains the termination status.
return parameters:	none
possible errors:	

Exit provides an exit from an active process.

system call: **.FACCESS**
number: 26h

purpose: This call tests file access.

summary:

```
hl   ->  path name
jsys    .faccess
c    =   access bits
```

calling parameters: hl The hl register pair points to the file path name.

return parameters: c The c register contains the applicable access bits set:

AC.READ	read
AC.EXEC	execute
AC.WRIT	write
AC.APND	append

possible errors: ?filaccess

system call: **.FCHSTAT**
number: 22h

purpose: The call changes the status of a file.

summary:

```
hl  ->  path name
c   =   status to change
d   =   value
e   =   mask
jsys   .fchstat
```

calling parameters:

hl The hl register pair points to the path name of the file.

c The c register contains the status type which is to be changed.

d The d register contains the new value of the specified status type.

e The e register contains a mask of the status bits which are to be changed:

AC.READ	read
AC.EXEC	execute
AC.WRIT	write
AC.APND	append

return parameters: none

possible errors: ?filaccess

Fchstat allowes the owner of a file to change file access privileges for the owner, group or others. Possible methods of access are:

read
write
append
execute
no access permitted

Any of these access methods may be combined. Please refer to the following page for a table of Fchstat calls.

Table of Fchstat Calls

Who*	C Register	Status Type	Location of New Information
p	ST.OWNER	owner	de = new value
p	ST.GROUP	group	de = new value
p&o	ST.AOWNER	access owner	d = new value, e = mask
p&o	ST.AGROUP	access group	d = new value, e = mask
p&o	ST.AOTHER	access other	d = new value, e = mask
p	ST.TCREATE	time created	de -> 6 byte buffer
p	ST.TMODIFY	time last modified	de -> 6 byte buffer
p	ST.TACCESS	time last accessed	de -> 6 byte buffer
p	ST.TDUMPED	time last dumped	de -> 6 byte buffer

*p = privileged user
o = owner

system call: **.FEXEC**
number: 4Bh

purpose: This call forks and executes a program.

summary:

```
bc   =   channel mask
de   ->  argument list
hl   ->  file pathname
jsys   .fexec
hl   =   child process number
```

calling parameters:

bc The bc register pair contains the 16 bit mask of channels which are to be passed to the program.

de The de register pair points to a list of pointers. The list of pointers is terminated by a null pointer (=0). Each of the pointers points to a null terminated character string. Each of the strings is an argument which is passed to the forked process.

hl The hl register pair points to the file path name.

return parameters: hl The hl register pair contains the child process number.

possible errors: ?badname

Fexec begins execution of a program and returns control to the user. This is similar to the Exec instruction except that a new process is created.

system call: **.FLINK**
number: 24h

purpose: This call establishes a link to a file.

summary:

```
hl  ->  old path name
de  ->  new path name
jsys   .flink
```

calling parameters:

hl — The hl register pair points to the existing (old) file path name for which a new link is to be established.

de — The de register pair points to the new file path name which is to be established.

return parameters: none

possible errors:

?badname
?isdir
?numlinks
?diraccess

system call: **.FSHELL**
number: 48h

purpose: This call forks a Shell process.

summary:

```
de  ->  argument list
jsys   .fshell
hl  =   process id
```

calling parameters: de — The de register pair points to a list of pointers. The list of pointers is terminated by a null pointer (=0). Each of the pointers points to a null terminated character string. Each of the strings is an argument which is passed to the forked process.

return parameters: hl — The hl register pair contains the process id number. This is a 16 bit value.

possible errors:

system call: **.FSTAT**
number: 20h

purpose: This call determines the status of a file.

summary:
```
de    ->   6/128 Byte Buffer
hl    ->   path name
c     =    desired information
jsys      .fstat
de    =    return value
hl    =    return value
```

calling parameters:

hl The hl register pair points to the path name of the file.

c The c register contains the request to the system for the desired information. Refer to the table below.

return parameters:

dehl The de (and in some cases the hl) register pair contains the requested information. Refer to the table below.

possible errors: ?badname

Fstat returns file status information. Please refer to the table of Fstat Calls on the following page.

Table of FSTAT Calls

C	Information	Location of
Register	Returned	Infomation
ST.ALL	all of inode	de -> 128 byte buffer
ST.OWNER	owner	de
ST.GROUP	group	de
ST.AOWNER	access owner	d
ST.AGROUP	access group	d
ST.AOTHER	access other	d
ST.FTYPE	file type	d = IS.ORDIN IS.DIRECT IS.CHAR IS.BLOCK e = device number
ST.SIZE	file size	dehl
ST.NLINKS	number of links	de
ST.INUM	inode number	de
ST.DEVICE	device number	d
ST.TCREATE	time created	de -> 6 byte buffer
ST.TMODIFY	time last modified	de -> 6 byte buffer
ST.TACCESS	time last accessed	de -> 6 byte buffer
ST.TDUMPED	time last dumped	de -> 6 byte buffer

system call: **.GETDATE**
number: 30h

purpose: This call gets the date.

summary:

```
jsys   .getdate
d   =   day
e   =   year
h   =   month
l   =   date
```

calling parameters: none

return parameters:

d The d register contains the day of the week where 1 represents Sunday, 2 represents Monday, etc.

e The e register contains the year minus 1900. This means that 1980 will be represented as 80 and 2004 will be 104.

h The h register contains the month where 1 represents January, 2 represents February, etc.

l The l register contains the day of the month in the range between 1 and 31 inclusive.

possible errors:

Getdate returns the current date as recorded by the Cromix system clock.

system call:	**.GETDIR**
number:	02h
purpose:	This call determines the current directory path name.
summary:	hl -> buffer jsys .getdir
calling parameters:	hl The hl register pair points to a 128 byte buffer for the current directory path name.
return parameters:	none
possible errors:	

Getdir returns the path name of the current directory.

system call: **.GETGROUP**
number: 36h

purpose: This call gets the group id.

summary

```
c    =   id type
jsys    .getgroup
hl   =   group id
```

calling parameters: c The c register contains a flag indicating the type of identification desired.

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM

return parameters: hl The hl register pair contains the type of group identification requested.

possible errors:

system call: **.GETMODE**
number: 12h

purpose: This call gets the characteristics of a console device.

summary:

```
b    =   channel
c    =   mode number
jsys   .getmode
d    =   return value
```

calling parameters:

b The b register contains the channel number of the device.

c The c register contains the mode which is to be tested. The c register may be loaded with one of the following:

C Register	Significance
MD.IBAUD	input speed
MD.OBAUD	output speed
MD.MODE1	mode1
MD.MODE2	mode2
MD.ERASE	auxilliary input erase character
MD.DLECHO	input delete echo character
MD.KILL	input line kill character
MD.SIGNAL	user input signal character
MD.WIDTH	output page width
MD.LENGTH	output page length
MD.BMARGIN	output bottom margin width
MD.CRNULLS	nulls output after a carriage return
MD.NLNULLS	nulls output after a new line
MD.TABNULLS	nulls output after a tab
MD.FFNULLS	nulls output after a form feed or a vertical tab
MD.STATUS	channel status
MD.IDENT	channel identification

return parameters: d

The d register contains the return value as specified below:

MD.IBAUD & MD.OBAUD

If the c register contains MD.IBAUD then the speed code for the input baud rate will be returned in the d register.
If the c register contains MD.OBAUD then the speed code for the output baud rate will be returned in the d register.

Speed Code	Baud Rate
B.HANGUP	(hang up dataphone)
B.50	50
B.75	75
B.110	110
B.134	134.5
B.150	150
B.200	200
B.300	300
B.600	600
B.1200	1200
B.1800	1800
B.2400	2400
B.4800	4800
B.9600	9600
B.EXTA	external A
B.EXTB	external B
B.19200	19200
B.AUTO	automatic *
B.NOCHG	no change

*automatic: Input carriage returns from keyboard are used to set the baud rate.

MD.MODE1

If the c register contains MD.MODE1 then the d register is returned with the bits set according to the options in effect.

Bit in D	Significance
MD1.HANGUP	hangup after last close
MD1.TAB	software tabs (expand as spaces)
MD1.LCASE	map upper to lower case on input
MD1.ECHO	echo (full duplex)
MD1.CR.NL	on input, map carriage return into new line (line feed) & echo new line or carriage return as carriage return-line feed
MD1.RAW	raw mode: wake up on all input characters
MD1.ODD	odd parity allowed on input
MD1.EVEN	even parity allowed on input

MD.MODE2

If the c register contains MD.MODE2 then the d register is returned with the bits set according to the options in effect.

Bit in D	Significance
MD2.PAUSE	after 24 lines output, wait for cntrl-Q
MD2.LATER	wait until character is used before echoing it
MD2.NOECNL	no echoing of line terminators
MD2.SGENABLE	user signal (MD.SIGCHAR) enable
MD2.ABENABLE	cntrl-C abort enable
MD2.FF	software formfeeds (expand as nls)
MD2.WRAP	software wrap-around (insert nl when page width (MD.WIDTH) is exceeded)

MD.ERASE

If the c register contains MD.ERASE then the value of the auxilliary input erase character is returned in the d register. This character may be used in addition to the control-H (H) and DELete characters to delete input characters from the specified device.

MD.DLECHO

If the c register contains MD.DLECHO then the value of the input delete echo character is returned in the d register ('R' or 'r' stands for Rubout or backspace-space-backspace).

MD.KILL

If the c register contains MD.KILL then the value of the input line kill character is returned in the d register.

MD.SIGNAL

If the c register contains MD.SIGNAL then the value of the user input signal character is returned in the d register.

MD.WIDTH

If the c register contains MD.WIDTH then the value of the output page width is returned in the d register.

MD.LENGTH

If the c register contains MD.LENGTH then the value of the output page length is returned in the d register.

MD.BMARGIN

If the c register contains MD.BMARGIN then the value of the output bottom Margin width is returned in the d register.

MD.CRNULLS
If the c register contains MD.CRNULLS then the number of nulls output after a carriage return is returned in the d register.

MD.NLNULLS
If the c register contains MD.NLNULLS then the number of nulls output after a new line is returned in the d register.

MD.TABNULLS
If the c register contains MD.TABNULLS then the number of nulls output after a tab is returned in the d register.

MD.FFNULLS
If the c register contains MD.FFNULLS then the number of nulls output after a form feed or a vertical tab is returned in the d register.

MD.STATUS
If the c register contains MD.STATUS then the channel status is returned in the d register.

Bit in d	
ST.CHARRDY	at least one character ready
ST.KEYBD	character entered since status last checked
ST.LINERDY	line ready
ST.SIGNAL	
ST.ABORT	

MD.IDENT

If the c register contains MD.IDENT then the channel identification is returned in the d register.

Bit in d	
ID.TTY	part of a dual I/O channel
ID.OUTPUT	output channel
ID.SERIAL	serial channel
ID.NOCHG	channel characteristics

cannot

be changed

```
system call:   .GETPOS
     number:   10h

    purpose:   This call gets the file pointer.

    summary:   b    =   channel number
               jsys   .getpos
               dehl=   file pointer


    calling
 parameters:   b    The b register contains the
                    channel number of the open
                    file.

     return
 parameters:   dehl The de and hl register pairs
                    contain the current value of
                    the file pointer.

   possible
     errors:   ?notopen
               ?notblk
```

Getpos returns the logical position byte value of the file pointer.

system call: **.GETPROC**
number: 3Ah

purpose: This call gets the process id.

summary:

```
jsys   .getproc
hl  =   process id
```

calling parameters: none

return parameters: hl The hl register pair contains the process id.

possible errors:

Getproc returns the process id of the caller's current active process

system call:	**.GETTIME**
number:	32h
purpose:	This call gets the time.

summary:	jsys .gettime
	e = hour
	h = minute
	l = second

calling parameters:	none
return parameters:	e The e register contains the hours portion of the current time based on a 24 hour clock (e.g., 6pm is represented by 18 hours). h The h register contains the minutes portion of the current time. This is the number of minutes since the current hour started. l The l register contains the seconds portion of the current time. This is the number of seconds since the current minute started.
possible errors:	

Gettime returns the current time as recorded by the Cromix system clock.

system call: **.GETUSER**
number: 34h

purpose: The call gets the user id.

summary:

```
c    =   idtype
jsys    .getuser
hl   =   user id
```

calling parameters: c The c register contains a flag indicating the type of identification desired:

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM

return parameters: hl The hl register pair contains the type of user identification requested.

possible errors: none

system call: **.INDIRECT**
number: 51h

purpose: This call executes the system call in a register.

summary:

```
a    =   call number
bc       according to system call
de       according to system call
hl       according to system call
jsys    .indirect
```

calling parameters: a — The a register contains the system call number.

return parameters:

possible errors:

system call: **.MAKDEV**
number: 00h

purpose: This call creates a new name for a device.

summary:

```
hl   ->  path name
d    =   type of device
e    =   device number
jsys   .makdev
```

calling parameters:

hl — The hl register pair points to the new path name for the device.

d — The d register indicates the type of device:

IS.BLOCK	block device
IS.CHAR	character device

e — The e register contains the device number.

return parameters: none

possible errors:

Makdev may be invoked only by a privileged user.

system call:	**.MAKDIR**	
number:	01h	
purpose:	This call creates a new directory.	
summary:	hl -> path name jsys .makdir	
calling parameters:	hl	The hl register pair points to the path name of the new directory.
return parameters:	none	
possible errors:		

Makdir is used to create a new directory.

```
system call:   .MOUNT
     number:   04h

    purpose:   This call enables access to a file
               system.

    summary:   hl  ->  dummy path name
               de  ->  block device path name
               c   =   type of access
               jsys   .mount


    calling
 parameters:   hl   The hl register pair points to
                    a buffer containing the path
                    name of the file system which
                    is to be mounted.

               de   The de register pair points to
                    a buffer containing the path
                    name of the block device on
                    which the file system is to be
                    mounted.

               c    The c register indicates the
                    desired access:

                         0   read/write
                         1   read only

     return
 parameters:   none

   possible
     errors:   device is inaccessible or not a
               device
               file system path name does not exist
               ?mttable
               ?fsbusy
```

Mount declares that a file system is to be mounted on a specified device. This is done by inserting the diskette containing the file system in a disk drive and giving it a file system path name. References to the file system path name will refer to the root file of the file system which was mounted.

The dummy path name with which mount is called will be the file system path name while the file system remains mounted. When the system is unmounted the name will revert to being a dummy path name.

system call: **.MULT**
number: 53h

purpose: multiply

summary:

```
bc  =   multiplier
hl  =   multiplicand
jsys   .mult
dehl=   product
```

calling parameters:

return parameters:

possible errors:

```
system call:   .PRINTF
     number:   1Bh

    purpose:   generate formatted output

    summary:   b    =   channel
               hl   ->  control string
               push     all arguments
               jsys    .printf
               pop      all arguments
```

calling parameters:

b — The b register contains the output channel number.

hl — The hl register pair points to the control string.

stack — All arguments to the printf call must be pushed onto the stack before the call and popped off of the stack after the call.

return parameters: none

possible errors:

The printf system call will output a formatted string to the file specified by the **b** register.

The null terminated control string is composed of regular characters and conversion specifications. Regular characters are just copied directly to the output file. Conversion specification characters are introduced by the percent (%) sign and terminated by the conversion character itself.

The conversion specification characters have the following format:

```
%-xxx.yyyLz
```

The percent sign and the conversion character itself are required, all of the conversion specification characters in between are optional.

A minus sign may follow the percent sign. If it is included, the argument will be left justified. Otherwise the argument will be right justified.

Following this may be two strings of digits separated by a period (represented by **xxx.yyy** above). The first of these numbers represents the minimum field width. If it is not included, the minimum field width is assumed to be zero. The second of these numbers represents the maximum field width. If it is not included, the maximum field width is undefined (large).

If the character **L** appears after this it signifies that the argument is a long (32 bit) number. If it is absent, the argument is assumed to be short (16 bits).

The conversion character itself (represented by **z** above) may be any one of the following:

d   The argument is converted to a decimal number.
u   The argument is converted to an unsigned decimal number.
x   The argument is converted to an unsigned hexadecimal number.
c   The argument is assumed to be a single character. When this argument is pushed onto the stack, the character must be in the low order byte of the register pair which is pushed.
s   The argument is assumed to be a character string. A pointer to this string must be pushed onto the stack in place of the string itself.

Example:

```
     ld       b,stdout
     ld       hl,number
     push     hl
     ld       hl,control
     jsys     .printf
     pop      hl
       .
       .
     number   defw 123
     control  defb 'This is a number: %d\\n\\0'
```

This program segment will display the following line on the terminal:

```
This is a number:  123
```

system call: **.OPEN**
number: 09h

purpose: This call opens a file for access.

summary:

```
hl  ->  path name
c   =   access mode
d   =   exclusive mode
jsys   .open
b   =   channel
```

calling parameters:

hl The hl register pair points to a buffer containing the path name of the file which is to be opened.

c The c register contains the access mode for opening the file. The following labels represent the values of the c register required in order to establish the desired access mode.

non-exclusive access

OP.READ	read only
OP.WRITE	write only
OP.RDWR	read/write
OP.APPEND	append

exclusive access

OP.XREAD	read only
OP.XWRITE	write only
OP.XWRITE	read/write
OP.XAPPEND	append

d The d register contains the mask for exclusive access. Each of the specified bits must be set to prevent the file from being opened by more than one user with the specified access. The bits may be anded together in order to set more than one bit.

exclusive access bits

^OP.READ	read only
^OP.WRITE	write only
^OP.RDWR	read/write
^OP.APPEND	append

return
parameters: b The b register contains the channel number which the system has assigned to the file.

possible
errors: ?filtable
?badname
?diraccess
?isdir

Open assigns a channel number to the specified file. The user is then allowed to read from and/or write to the file.

system call: **.RDBYTE**
number: 16h

purpose: This call reads a byte.

summary:

```
b     =   channel
jsys    .rdbyte
a     =   byte
```

calling parameters: b — The b register contains the channel number of the file.

return parameters: a — The a register contains the byte which was read.

possible errors:
?notopen
?filaccess
?ioerror
?endfile

The next sequential byte (reading toward the end of the file) is read from the file which is open on the specified channel.

```
system call:   .RDLINE
     number:   18h

    purpose:   This call reads a line.

    summary:   de   =   maximum bytes
               hl   ->  buffer
               b    =   channel
               jsys    .rdline
               de   =   bytes read


    calling
 parameters:   de    The de register pair contains
                     the maximum number of bytes
                     which are to be read by this
                     call.

               hl    The hl register pair points to
                     the buffer in which the line is
                     to be returned.

               b     The b register contains the
                     channel number of the file.

     return
 parameters:   de    The de register pair contains
                     the actual number of bytes
                     read.

   possible
     errors:   ?notopen
               ?filaccess
               ?ioerror
               ?endfile
```

A number of sequential bytes (reading toward the end of file) is read from the file which is open on the specified channel.

The buffer is filled with bytes until an end of line indicator is encountered (line feed or null character). A maximum of 512 characters may be read.

system call: **.RDSEQ**
number: 14h

purpose: This call reads sequentially.

summary:

```
de   =    byte count
hl   ->   buffer
b    =    channel
jsys    .rdseq
de   =    number of bytes read
```

calling parameters:

de The register pair contains the number of sequential bytes to be read starting from the current position of the file pointer.

hl The hl register pair points to the buffer in which bytes are to be returned.

b The b register contains the channel number of the file which is to be read.

return parameters:

de The de register contains the actual number of bytes read.

possible errors:

?notopen
?filaccess
?ioerror
?endfile

The next sequential specified number of bytes (reading toward the end of the file) is read from the file which is open on the specified channel.

system call: **.SETDATE**
number: 31h

purpose: The call sets the date.

summary:

```
e    =   year
h    =   month
l    =   date
jsys   .setdate
```

calling parameters:

e — The e register contains the year minus 1900. This means that 1980 will be represented as 80 and 2004 will be 104.

h — The h register contains the month where 1 represents January, 2 represents February, etc.

l — The l register contains the day of the month in the range between 1 and 31 inclusive.

return parameters: none

possible errors:

Setdate sets the Cromix system clock. Note that the parameters are binary numbers.

Setdate may be invoked only by the privileged user.

system call:	**.SETDIR**
number:	03h
purpose:	This call changes the current directory.
summary:	hl -> path name jsys .setdir
calling parameters:	hl The hl register pair points to the new directory path name.
return parameters:	none
possible errors:	?notdir ?diraccess

Setdir changes the current directory to the one which is specified.

system call: **.SETGROUP**
number: 37h

purpose: This call sets the group id.

summary:

```
b    =   type of id to change
c    =   new id label
hl   =   new id number
jsys   .setgroup
```

calling parameters:

b The b register contains the type of id which is to be changed. Refer to the following table.

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM

c The c register is used to indicate the value of the id type specified by the b register. This value may be the value of one of the other id types or the value specified by the hl register:

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM
ID.HL

hl If the c register contains ID.HL then the hl register pair must contain a 16 bit id number.

return parameters: none

possible errors:

Setgroup changes the group id of the current process to that which is specified. This call may be invoked only by a privileged user when the c register has the value of ID.HL.

system call: **.SETMODE**
number: 13h

purpose: This call sets the characteristics of a character device.

summary:

```
b     =   channel
c     =   mode number
d     =   new value
e     =   mask
jsys     .getmode
d     =   old value
```

calling parameters:

b The b register contains the channel number of the device.

c The c register contains the mode which is to be set. The c register may be loaded with one of the following:

C Register	Significance
MD.IBAUD	input speed
MD.OBAUD	output speed
MD.MODE1	mode1
MD.MODE2	mode2
MD.ERASE	auxilliary input erase character
MD.DLECHO	input delete echo character
MD.KILL	input line kill character
MD.SIGNAL	user input signal character
MD.WIDTH	output page width
MD.LENGTH	output page length
MD.BMARGIN	output bottom margin width
MD.CRNULLS	nulls output after a carriage return
MD.NLNULLS	nulls output after a new line
MD.TABNULLS	nulls output after a tab
MD.FFNULLS	nulls output after a form feed or a vertical tab

d The d register contains the new value as specified below:

MD.IBAUD & MD.OBAUD

If the c register contains MD.IBAUD then d must be set to the value of the desired input baud rate using the speed code listed below.
If the c register contains MD.OBAUD then d must be set to the value of the desired output baud rate using the speed code listed below.

Speed Code	Baud Rate
B.HANGUP	(hang up dataphone)
B.50	50
B.75	75
B.110	110
B.134	134.5
B.150	150
B.200	200
B.300	300
B.600	600
B.1200	1200
B.1800	1800
B.2400	2400
B.4800	4800
B.9600	9600
B.EXTA	external A
B.EXTB	external B
B.19200	19200
B.AUTO	automatic *
B.NOCHG	no change

*automatic: Input carriage returns from keyboard are used to set the baud rate.

MD.MODE1

If the c register contains MD.MODE1 then the e register acts as a mask and the d register is used to indicate the desired value of the bits specified in the E register.

Bit in D	Significance
MD1.HANGUP	hangup after last close
MD1.TAB	software tabs (expand as spaces)
MD1.LCASE	map upper to lower case on input
MD1.ECHO	echo (full duplex)
MD1.CR.NL	on input, map carriage return into new line (line feed) & echo new line or carriage return as carriage return-line feed
MD1.RAW	raw mode: wake up on all characters
MD1.ODD	odd parity allowed on input
MD1.EVEN	even parity allowed on input

MD.MODE2

If the c register contains MD.MODE2 then the e register acts as a mask and the d register is used to indicate the desired value of the bits specified in the E register.

Bit in D	Significance
MD2.PAUSE	after 24 lines output, wait for cntrl-Q
MD2.ECHIN	always echo on interrupt input
MD2.NOECNL	no echoing of line terminators
MD2.SGENABLE	user signal (MD.SIGCHAR) enable
MD2.ABENABLE	cntrl-C abort enable
MD2.FF	software formfeeds (expand as nls)
MD2.WRAP	software wrap-around (insert nl when page width (MD.WIDTH) exceeded)

MD.ERASE

If the c register contains MD.ERASE then the d register is used to indicate the desired value of the auxilliary input erase character. This character may be used in addition to the control-H (H) and delete characters to delete input characters from the specified device.

MD.DLECHO

If the c register contains MD.DLECHO then the d register is set to indicate the desired value of the input delete echo character ('R' or 'r' stands for Rubout or backspace-space-backspace).

MD.KILL

If the c register contains MD.KILL then the d register is set to indicate the desired value of the input line kill character.

MD.SIGNAL

If the c register contains MD.SIGNAL then the d register is set to indicate the desired value of the user input signal character.

MD.WIDTH

If the c register contains MD.WIDTH then the d register is set to indicate the desired value of the output page width (1...256).

MD.LENGTH

If the c register contains MD.LENGTH then the d register is set to indicate the desired value of the output page length (1...256).

MD.BMARGIN

If the c register contains MD.BMARGIN then the d register is set to indicate the desired value of the output bottom margin width (0...255).

MD.CRNULLS
If the c register contains MD.CRNULLS then the d register is set to indicate the desired number of nulls output after a carriage return (0...).

MD.NLNULLS
If the c register contains MD.NLNULLS then the d register is set to indicate the desired number of nulls output after a new line (0...).

MD.TABNULLS
If the c register contains MD.TABNULLS then the d register is set to indicate the desired number of nulls output after a tab.

MD.FFNULLS
If the c register contains MD.FFNULLS then the d register is set to indicate the desired number of nulls output after a form feed.

return parameters:

possible errors:

system call: **.SETPOS**
number: 11h

purpose: This call sets the file pointer.

summary:

```
b    =   channel
dehl=   file pointer
c    =   mode
jsys   .setpos
```

calling parameters:

b The b register contains the channel number of an open file.

dehl The de and hl register pairs contain the desired change in position of the file pointer.

c The c register contains the mode. This is the location from which and direction in which the position of the file pointer is established.

POS.BEGIN	forward from the beginning of file
POS.CURRENT	forward from the current position
POS.END	forward from the end of file
-POS.CURRENT	backward from current position
-POS.END	backward from end of file

return parameters: none

possible errors: ?notblk
?filaccess

Setpos positions the file position pointer to the logical byte position specified.

system call: **.SETTIME**
number: 33h

purpose: This call sets the time.

summary:

```
e    =   hour
h    =   minute
l    =   second
jsys   .settime
```

calling parameters:

e — The e register contains the hours portion of the current time based on a 24 hour clock (e.g., 6pm is represented by 18 hours).

h — The h register contains the minutes portion of the current time. This is the number of minutes since the current hour started.

l — The l register contains the seconds portion of the current time. This is the number of seconds since the current minute started.

return parameters: none

possible errors:

Settime sets the Cromix system clock. Note that the parameters are binary numbers.

Settime may be invoked only by the privileged user.

system call: **.SETUSER**
number: 35h

purpose: This call changes the user id.

summary:

```
b    =   type of id to change
c    =   new id type
hl   =   new id number
jsys   .setuser
```

calling parameters:

b The b register contains the type of id which is to be changed:

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM

c The c register is used to indicate the value of the id type specified by the b register. This value may be the value of one of the other id types or the value specified by the hl register:

ID.EFFECTIVE
ID.LOGIN
ID.PROGRAM
ID.HL

hl If the c register contains ID.HL then the hl register pair must contain a 16 bit id number.

return parameters: none

possible errors:

Setuser changes the id of the current process to that which is specified. This call may be invoked only by a privileged user when the c register has the value of ID.HL.

system call:	**.SHELL**	
number:	49h	
purpose:	This call transfers to a Shell process.	
summary:	de -> argument list jsys .shell	
calling parameters:	de	The de register pair points to a list of pointers. The list of pointers is terminated by a null pointer (=0). Each of the pointers points to a null terminated character string. Each of the strings is an argument which is passed to the forked process.
return parameters:		
possible errors:		

```
system call:   .TRUNC
     number:   0Dh

    purpose:   This call truncates an open file.

    summary:   b    =   channel
               jsys   .trunc


    calling
 parameters:   b    The b register contains the
                    channel number of the open
                    file.

     return
 parameters:

   possible
     errors:   ?notopen
```

Trunc deletes the part of a file which is past the current position of the file pointer through the end of the file.

system call:	**.UNMOUNT**
number:	05h
purpose:	This call disables access to a file system.
summary:	hl -> block device path name jsys .unmount
calling parameters:	hl The hl register pair points to a buffer containing the path name of the block device which is to be unmounted.
return parameters:	none
possible errors:	?notmount ?fsbusy

Used in conjunction with mount, unmount declares that the device no longer has the previously specified file system.

When the system is unmounted the file system path name will revert to being a dummy path name.

system call:	**.UPDATE**
number:	52h
purpose:	update disk buffers
summary:	jsys .update
calling parameters:	none
return parameters:	none
possible errors:	?ioerror

Update causes all open files to be updated with the current contents of their buffers. This is also done automatically upon closing a file.

system call: **.VERSION**
number: 55h

purpose: get system version number

summary:

```
jsys   .version
hl  =   version number
```

calling parameters: none

return parameters: hl The hl register pair contains the Cromix Operating System version number.

possible errors: none

system call: **.WAIT**
number: 45h

purpose: This call waits for the termination of a child process.

summary:

```
c    =   conditional flag
hl   =   process id number
jsys   .wait
hl   =   child process number
de   =   process termination status
c    =   system termination status
```

calling parameters:

c — If the c register equals zero, this call will not return until a child process has terminated.

If the c register equals one, this call will return immediately. An error will be returned if no child process has terminated.

hl — If the hl register pair contains a zero, this call will wait for the termination of any child process.

If the hl register pair is set equal to a process id number (PID), this call will wait for the termination of the specified process.

return parameters:

hl — The hl register pair contains the child process number.

de — The de register pair contains the process termination status which is returned by jsys .exit.

c — The c register contains the system termination status.

possible errors: ?nochild

Wait informs the parent process when a child process is no longer active.

```
system call:   .WRBYTE
     number:   17h

    purpose:   This call writes a byte.

    summary:   b    =   channel
               a    =   byte
               jsys   .wrbyte


    calling
 parameters:   b    The b register contains the
                    channel number of the file.

               a    The a register contains the
                    byte which is to be written.

     return
 parameters:   none

   possible
     errors:   ?notopen
               ?filaccess
               ?ioerror
```

A byte is written to the file which is open on the specified channel. The byte will be written just after the last byte which was written since the file was last open. Note that this may over-write information which was written to the file when it was previously open.

system call: **.WRLINE**
number: 19h

purpose: This call writes a line.

summary:

```
hl  ->  buffer
b   =   channel
jsys   .wrline
de  =   bytes written
```

calling parameters:

hl   The hl register pair points to the buffer in which the line to be written is stored.

b   The b register contains the channel number of the file.

return parameters:

de   The de register pair contains the number of bytes written.

possible errors:

?notopen
?filaccess
?ioerror

A series of sequential bytes is written to the file which is open on the specified channel. The bytes will be written just after the last byte which was written since the file was last open.

Bytes are written until an end of line indicator is encounterd (line feed or null character). A maximum of 512 bytes may be written.

Note that this may over-write information which was written to the file when it was previously open.

system call: **.WRSEQ**
number: 15h

purpose: This call writes sequentially.

summary:

```
de   =   byte count
hl   ->  buffer
b    =   channel
jsys   .wrseq
de   =   bytes written
```

calling parameters:

de — The de register pair contains the number of sequential bytes to be written starting from the current position of the file pointer.

hl — The hl register pair points to the buffer in which the bytes to be written are stored.

b — The b register contains the channel number of the file.

return parameters:

de — The de register pair contains the actual number of bytes written. If this is not equal to the value of de as a calling parameter an error has occurred.

possible errors:

?notopen
?filaccess
?ioerror

A series of sequential bytes is written to the file which is open on the specified channel. The bytes will be written just after the last byte which was written since the file was last open. Note that this may over-write information which was written to the file when it was previously open.

Error Conditions

If the Cromix Operating System cannot complete a system call in the normal manner, an error will be generated. The Operating System flags an error condition by setting the carry bit in the flag register (the carry flag). A normal return from a system call is indicated by a reset carry flag.

If an error has occurred (carry flag is set or is equal to one), the **a** register will contain the error code. The type of error which was returned may be established by comparing the **a** register with the following list of error codes. Each error code is preceded by the error number.

1 **?badchan** An invalid channel number was specified. The Operating System must be called with a channel number which was assigned at the time a file was opened or created.

2 **?toomany** All possible channels already open.

3 **?notopen** The specified channel has not been opened or was closed prior to the system call. A file must be opened (using the .open or .create call) prior to being accessed for I/O.

4 **?endfile** An end of file condition exists on the file being processed. There is no data in the file beyond (in a forward direction) the current file position.

5 **?ioerror** A physical data transmission error has occurred.

6 **?filtable** The file table has been exhausted.

7 **?notexist** The specified file does not exist. Make sure that the path name properly identifies the desired file.

8 **?badname** The file name which was specified does not conform to proper file name syntax. The name is too long or contains illegal characters.

9 **?diraccess** An attempt has been made to access a directory which the current user may not access. Make sure the path name does not include any directories with priviliged access.

10 **?filaccess** An attempt has been made to access a file which the current user may not access.

11 **?exists** An attempt has been made to create a file which already exists.

12 **?nospace** An attempt has been made to write to a full disk.

13 **?noinode** No inodes are left.

14 **?inotable** The inode table is exhausted.

15 **?badcall** The system call which was specified is illegal.

16 **?filsize** The size of the file is too big.

17 **?mnttable** The mount table is exhausted.

18 **?notdir** The specified path name was not that of a directory.

19 **?isdir** The specified path name is that of a directory.

20 **?priv** An attempt was made to invoke a privileged system call by other than a privileged user.

21 **?notblk** The specified device is not a block special device.

22 **?fsbusy** The requested file system was busy.

23 **?notordin** The requested file is not an ordinary file.

24 **?notmount** The specified device was not mounted prior to the call.

25 **?nochild** No child process.

26 **?nomemory** There is not enough memory.

27 **?ovflo** An overflow occurred during a divide operation.

28 **?argtable** The argument table is exhausted.

29 **?arglist** The argument list which was provided is incorrect.

30 **?numlinks** This operation would have created too many links to the specified file or device.

31 **?difdev** Cross device link. File references cannot exist across disks.

32 **?nodevice** No device driver for referenced device.

33 **?usrtable** The user process table is exhausted.

34 **?badvalue** The specified value was out of range.

35 **?notconn** The requested I/O device was not connected to the system.

36**keybaud** The baud rate must be set from the keyboard.

37 **?diruse** An attempt was made to delete a directory which was in use. All files must be deleted from a directory before it may be deleted.

38 **?filuse** The requested file is an exclusive access file and was in use.

39 **?nomatch** There was no match on the specified ambiguous path name.

40 **?chnaccess** Channel access

41 **?notcromix** The specified disk is not compatible with the Cromix Operating System.

Appendix

EQUates for Cromix System Calls and Labels Used in This Manual

```
stdin       equ  0      ;standard input channel
stdout      equ  1      ;standard output channel
stderr      equ  2      ;standard error channel

argc        equ  40H    ;location for argument count
argv        equ  42H    ;location for argument list vector
arg0        equ  0      ;arg offset
arg1        equ  2      ;arg offset
arg2        equ  4      ;arg offset
arg3        equ  6      ;arg offset
arg4        equ  8      ;arg offset

;
; C-register modes for .create, .open
;
op.read     equ  0      ;read only
op.write    equ  1      ;write only
op.rdwr     equ  2      ;read and write
op.append   equ  3      ;append only
op.xread    equ  4      ;exclusive read only
op.xwrite   equ  5      ;exclusive write only
op.xrdwr    equ  6      ;exclusive read and write
op.xappend  equ  7      ;exclusive append only

op.truncf   equ  80H    ;truncate on create flag
op.condf    equ  40H    ;conditional create flag

;
; C-register modes for .fstat, .cstat, .fchstat, .cchstat
;
st.all      equ  0      ;all of inode (128 bytes)
st.owner    equ  1      ;de = owner
st.group    equ  2      ;de = group
st.aowner   equ  3      ;d = owner access, e = mask
st.agroup   equ  4      ;d = group access, e = mask
st.aother   equ  5      ;d = other access, e = mask
st.ftype    equ  6      ;d = file type, e = special device #
st.size     equ  7      ;dehl = file size
st.nlinks   equ  8      ;de = number of links
st.inum     equ  9      ;de = inode number
st.device   equ  10     ;d = device containing inode
st.tcreate  equ  11     ;de-> time created
st.tmodify  equ  12     ;de-> time last modified
st.taccess  equ  13     ;de-> time last accessed
st.tdumped  equ  14     ;de-> time last dumped
;
; file types for st.ftype
```

```
;
is.ordin        defl    0       ;ordinary file
is.direct       defl    1       ;directory file
is.char         defl    2       ;character device
is.block        defl    3       ;block device
;
; access bits for access flags
;
ac.read         defl    0       ;read access bit
ac.exec         defl    1       ;execute access bit
ac.writ         defl    2       ;write access bit
ac.apnd         defl    3       ;append access bit
;
; C-register modes for .setuser, .getuser, .setgroup, .getgroup
;
id.effective    equ     0       ;effective id
id.login        equ     1       ;login id
id.program      equ     2       ;program id
id.hl           equ     3       ;id contained in HL register
```

```
; SYSTEM CALL NUMBERS

.makdev     equu  00H    ;makdev(d,e,hl)         make device entry
.makdir     equ   01H    ;makdir(hl)             make a directory
.getdir     equ   02H    ;getdir(hl)             get current directory name
.setdir     equ   03H    ;setdir(hl)             change current directory

.mount      equ   04H    ;mount(c,de,hl)         mount file system
.unmount    equ   05H    ;unmount(hl)            unmount file system
.delete     equ   06H    ;delete(hl)             delete file
.chkdev     equ   07H    ;chkdev(d,e)            check for device driver

.create     equ   08H    ;b=create(c,hl)         create & open file
.open       equ   09H    ;b=open(c,hl)           open file
.chdup      equ   0AH    ;c=chdup(b)             duplicate channel
.close      equ   0BH    ;close(b)               close file

.trunc      equ   0DH    ;trunc(b)               truncate open file

.getpos     equ   10H    ;dehl=getpos(b)         get file position
.setpos     equ   11H    ;setpos(c,dehl)         set file position
.getmode    equ   12H    ;d=getmode(b,c)         get device characteristics
.setmode    equ   13H    ;d=setmode(b,c,d,e)     set device characteristics

.rdseq      equ   14H    ;de=rdseq(b,de,hl)      read n bytes
.wrseq      equ   15H    ;de=wrseq(b,de,hl)      write n bytes
.rdbyte     equ   16H    ;a=rdbyte(b)            read 1 byte
.wrbyte     equ   17H    ;wrbyte(b,a)            write 1 byte
.rdline     equ   18H    ;de=rdline(b,de,hl)     read a line
.wrline     equ   19H    ;de=wrline(b,hl)        write a line
;
.printf     equ   1BH    ;printf(b,hl)           print formatted string
.error      equ   1CH    ;error(a,b,de,hl)       print error message

.fstat      equ   20H    ;fstat(c,de,hl)         get file status (inode)
.cstat      equ   21H    ;cstat(b,c,de)          get channel status (inode)
.fchstat    equ   22H    ;fchstat(c,de,hl)       change file status
.cchstat    equ   23H    ;cchstat(b,c,de)        change channel status
.flink      equ   24H    ;flink(de,hl)           link to file
.clink      equ   25H    ;clink(b,de)            link to open channel
.faccess    equ   26H    ;faccess(c,hl)          test file access
.caccess    equ   27H    ;caccess(b,c)           test channel access

.getdate    equ   30H    ;d,e,h,l=getdate()      get date
.setdate    equ   31H    ;setdate(e,h,l)         set date
.gettime    equ   32H    ;e,h,l=gettime()        get time
.settime    equ   33H    ;settime(e,h,l)         set time

.getuser    equ   34H    ;de,hl=getuser()        get user id
.setuser    equ   35H    ;setuser(hl)            set user id
.getgroup   equ   36H    ;de,hl=getgroup()       get group id
.setgroup   equ   37H    ;setgroup(hl)           set group id
```

```
.getproc    equ  3AH    ;hl=getproc()              get process id

.wait       equ  45H    ;c,de,hl=wait()            wait for child process
.exit       equ  46H    ;exit(hl)                  exit process (close files)

.fshell     equ  48H    ;fShell(de)                fork a Shell process
.shell      equ  49H    ;Shell(de)                 transfer to Shell process
.fexec      equ  4BH    ;fexec(bc,de,hl)           fork and execute program
.exec       equ  4CH    ;exec(bc,de,hl)            execute program

.update     equ  52H    ;update()                  update disk I/O buffers
.mult       equ  53H    ;dehl=mult(bc,hl)          multiply
.divd       equ  54H    ;de,hl=divd(dehl,bc)       divide
.version    equ  55H    ;hl=version()              get system version #
```

```
; .SETMODE & .GETMODE call numbers (to be loaded into the c-register)
;
md.ibaud        equ     0
md.obaud        equ     1
md.model        equ     2
md.mode2        equ     4
md.erase        equ     6
md.dlecho       equ     7
md.kill         equ     8
md.signal       equ     9
md.width        equ     11
md.length       equ     10
md.bmargin      equ     12
md.nlnulls      equ     15
md.tabnulls     equ     16
md.ffnulls      equ     17
md.crnulls      equ     18
md.status       equ     13
md.ident        equ     14

; byte contents of the d-register for md.ibaud & md.obaud calls:
b.hangup        equ     0
b.50            equ     1
b.75            equ     2
b.110           equ     3
b.134           equ     4
b.150           equ     5
b.200           equ     6
b.300           equ     7
b.600           equ     8
b.1200          equ     9
b.1800          equ     10
b.2400          equ     11
b.4800          equ     12
b.9600          equ     13
b.exta          equ     14
b.extb          equ     15
b.19200         equ     16
b.auto          equ     17

b.nochg         equ     255
```

```
; bits of the d- & e-registers for md.model calls:
; (the bits set in the e-register determine which bits are changed,
;  the bits of the d-register determine whether the bits are set or reset.)
mdl.hangup      equ     0
mdl.tab         equ     1
mdl.lcase       equ     2
mdl.echo        equ     3
mdl.cr.nl       equ     4
mdl.raw         equ     5
mdl.odd         equ     6
mdl.even        equ     7

; bits of the d- & e-registers for md.mode2 calls:
; (the bits set in the e-register determine which bits are changed,
;  the bits of the d-register determine whether the bits are set or reset.)
md2.pause       equ     0
md2.later       equ     1
md2.noecnl      equ     2
md2.sgenable    equ     3
md2.abenable    equ     4
md2.ff          equ     5
md2.wrap        equ     6

; masks of legal model & mode2 bits for different types of channels:
mdlv.tty:       equ     0ffh
md2v.tty:       equ     0ffh
mdlv.outp:      equ     ^mdl.tab
md2v.outp:      equ     ^md2.ff+^md2.wrap
mdlv.inp:       equ     ^mdl.lcase+^mdl.echo+^mdl.cr.nl+
                        ^mdl.raw+^mdl.odd+^mdl.even
md2v.inp:       equ     ^md2.echin+^md2.noecnl+^md2.sgenable+^md2.abenable

; bits of the d-register for md.status calls:
st.charrdy      equ     0
st.keybd        equ     2
st.linerdy      equ     7
st.signal       equ     5
st.abort        equ     6

;bits of the d-register for md.ident calls:
id.tty          equ     0
id.output       equ     1
id.serial       equ     2
id.nochg        equ     3
```

List of Device Names and Numbers

Large Floppies	FDA	B	1
	FDB	B	2
	FDC	B	3
	FDD	B	4
Small Floppies	SFDA	B	5
	SFDB	B	6
	SFDC	B	7
	SFDD	B	8
Hard Disks	HD0	B	9
	HD1	B	10
	HD2	B	11
	HD3	B	12
	HD4	B	13
	HD5	B	14
	HD6	B	15
	HD7	B	16
Consoles	TTY1	C	1
	TTY2	C	2
	TTY3	C	3
	TTY4	C	4
	TTY5	C	5
	TTY6	C	6
Dot Matrix Printer	LPT1	C	25
Fully Formed Character Printer	TYP1	C	29